NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SIMULATING CANDIDATE MISSIONS FOR A NOVEL GLIDER UNMANNED UNDERWATER VEHICLE

by

John M. Seguin

March 2007

Thesis Advisor:	Don Brutzman
Second Readers:	Ray Jones
	Richard Williams

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY	2. REPORT DATE March 2007	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Simulating Candidate Missions for a Novel Glider Unmanned Underwater Vehicle		**5. FUNDING NUMBERS**	
6. AUTHOR: John M. Seguin			
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME AND ADDRESS N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT

Unmanned Underwater Vehicles (UUVs) are becoming ubiquitous in the framework of U.S. Navy operations. According to the U.S. Navy's UUV Master Plan (2004), research and development will expand UUV capabilities that enable diverse roles from Intelligence, Surveillance, and Reconnaissance (ISR) and Mine Countermeasures to Anti-Submarine Warfare (ASW) and Information Operations (IO). However, typical UUVs are severely limited in operational characteristics such as endurance and range which prevents their use conducting certain missions.

A novel UUV is currently being designed that is projected to support significantly greater endurance and range characteristics. This UUV is called Seadiver and is being designed by Institute of Engineering Science of Toulon, France with support from Naval Postgraduate School. It is a low-cost glider UUV which generates propulsion not with propellers or jet pumps, but rather by controlling its buoyancy. This method of propulsion is quite efficient and maybe capable of autonomous operation up to 30 days with a range of around 700 nautical miles. A UUV with such endurance and range exposes military missions previously impractical for UUVs especially when used in concert as an array of many UUVs.

This thesis creates a simulation using NPS-produced software simulation tools Simkit, Viskit and AUV Workbench that analyzes the capabilities and effectiveness of Seadiver UUVs conducting missions of tactical interest.

14. SUBJECT TERMS Discrete Event Simulation, Simkit, Diskit, Viskit, Seadiver, UUV, AUVW, Java, XML, Distributed Interactive Simulation, DIS			**15. NUMBER OF PAGES** 140
			16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

SIMULATING CANDIDATE MISSIONS FOR A NOVEL GLIDER UNMANNED UNDERWATER VEHICLE

John M. Seguin
Lieutenant, United States Navy
B.S., Old Dominion University, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS AND OPERATIONS

from the

NAVAL POSTGRADUATE SCHOOL
March 2007

Author: John M. Seguin

Approved by: Don Brutzman
               Thesis Advisor

               Ray Jones, RADM USN (Ret.)
               Second Reader

               Richard Williams, RADM USN (Ret.)
               Second Reader

               Dan C. Boger
               Chairman, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Unmanned Underwater Vehicles (UUVs) are becoming ubiquitous in the framework of U.S. Navy operations. According to the U.S. Navy's UUV Master Plan (2004), research and development will expand UUV capabilities that enable diverse roles from Intelligence, Surveillance, and Reconnaissance (ISR) and Mine Countermeasures to Anti-Submarine Warfare (ASW) and Information Operations (IO). However, typical UUVs are severely limited in operational characteristics such as endurance and range which prevents their use conducting certain missions.

A novel UUV is currently being designed that is projected to support significantly greater endurance and range characteristics. This UUV is called Seadiver and is being designed by Institute of Engineering Science of Toulon, France with support from Naval Postgraduate School. It is a low-cost glider UUV which generates propulsion not with propellers or jet pumps, but rather by controlling its buoyancy. This method of propulsion is quite efficient and maybe capable of autonomous operation up to 30 days with a range of around 700 nautical miles. A UUV with such endurance and range exposes military missions previously impractical for UUVs especially when used in concert as an array of many UUVs.

This thesis creates a simulation using NPS-produced software simulation tools Simkit, Viskit and AUV Workbench that analyzes the capabilities and effectiveness of Seadiver UUVs conducting missions of tactical interest.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

xii

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACRONYMS AND ABBREVIATIONS

2D	Two dimension
3D	Three dimension
API	Application Programming Interface
AUVW	Autonomous Underwater Vehicle Workbench
CSS	Continuous Systems Simulation
DIS	Distributed Interactive Simulation
DES	Discrete Event Simulation
DOM	Document Object Model
GUI	Graphical User Interface
HTML	Hypertext Markup Language
i	Conditional Statement in Event Graph Notation
ISITV	The Institute of Engineering Science of Toulon, France
JAXB	Java Architecture for XML Binding
M&S	Modeling and Simulation
NPS	Naval Postgraduate School
PCL	Property Change Listener
SGML	Standard Generalized Markup Language
t	Time Delay
UUV	Unmanned Underwater Vehicle
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language for Transformations

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Many people assisted in this research. Below is a list of the most important influences during my time at NPS creating this thesis. I would like to personally thank each and every one of you for your time and patience.

- To my loving wife Chrisanne, daughter Sophia, and son Tristan who sacrificed their time enabling my success during this project.

- Thesis advisor, Don Brutzman LCDR USN (ret), for his guidance and support during this thesis research.

- Second readers, Ray Jones RADM USN (ret) and Richard Williams RADM USN (ret), for their insight and time.

- Rick Goldberg, Mike Bailey, Terry Norbraten, Arnold Buss, and Jeff Weekley for providing advice, tools, and solutions to numerous aspects of this project.

- To my fellow curriculum students, especially Russ Schuhart and Steve Milgazo, for making this experience more enjoyable through their company and conversations.

- To my curriculum Program Officer, Col. Karl Pfeiffer, USAF and Program Advisor, Steve Iatrou for their support and encouragement.

THIS PAGE INTENTIONALLY LEFT BLANK

I.   INTRODUCTION

A.   OVERVIEW

This thesis supports the ongoing design and development of the Seadiver Unmanned Underwater Vehicle (UUV) at Naval Postgraduate School (NPS). Seadiver is a unique UUV in a relatively new class of underwater vehicles called gliders. Gliders have a unique set of characteristics such as high endurance and variable payload that might allow it to perform missions not suitable for UUVs in the past.

Capabilities of complex machines such as autonomous vehicles cannot be fully known prior to field tests. They can only be projected based on design and intent. Simulation can reduce that uncertainty inherent in design by virtually testing capabilities and configurations against environmental constraints prior to construction completion. This thesis creates a high fidelity simulation consisting of the Seadiver UUV and related entities, in combination with mission rehearsal and statistical analysis tools, to predict the value and accuracy of projected missions.

B.   PROBLEM STATEMENT

This thesis creates simulations based on projected missions of the Seadiver UUV for the purpose of answering the following questions in an attempt to validate the designed capabilities of Seadiver. The research questions are:

- Can a Discrete Event Simulation (DES) be constructed with the Simkit and Viskit tools to simulate these missions?

- Can high-fidelity simulation be used to visualize and validate the new missions for probability of success and provide insight into other advanced uses of a glider UUV?

- Can missions be validated as physically realistic using AUVW?

- Is operator control of numerous robots feasible using AUVW?

- What are the tactical capabilities of the Glider UUV, and what new missions are exposed by these novel capabilities?

C.   MOTIVATION

The interest and growing use in the U.S. military for unmanned systems is high and growing at an increasing rate. Recent uses in unmanned vehicles by the military are demonstrating their ability establish and maintain maritime superiority.   UUVs are attractive over other methods because they are force multipliers and risk-reducing agents. Additionally, they can be cost effective and their novel capabilities enable unique mission capabilities. The U.S. Navy's UUV Master Plan published in 2004 lists nine missions that support Sea Power 21 strategy. These missions are:

- Intelligence, Surveillance, and Reconnaissance (ISR)
- Mine Countermeasures (MCM)
- Anti-Submarine Warfare (ASW)
- Inspection / Identification
- Oceanography
- Communication / Navigation Network Nodes (CN3)
- Payload Delivery
- Information Operations (IO)
- Time Critical Strike (TCS)

In the realm of Undersea Warfare (USW), there are a number of available or upcoming UUVs that are being designed to fill these missions. The most significant limitation with UUVs is typically the power source. Most currently use batteries as their source of power and hence are severely limited. This is because as the size of the UUV increases, the amount of power required for propulsion increases at a greater proportion. As more batteries are installed into the UUV to extend its range, the power requirement also increases and quickly becomes an untenable situation. Therefore, the bottom line is that UUVs are severely limited in size, speed, and endurance due to the aforementioned relationship.

At Naval Postgraduate School (NPS), a UUV is currently being designed that will partially uncouple that relationship. It is a type of UUV known as a glider. Gliders are a recent innovation and work on the principle that changes in buoyancy create forward motion through the water. Implementations such as this are energy efficient allowing for endurance times on the order of weeks compared to hours for a typical UUV.

Additionally, what makes this glider unique is that most of its internal capacity is free-flood. This allows for a vehicle of arbitrary size with other aspects such as endurance and speed remaining constant, thus allowing for large payloads and an expanded range of missions for which it is suited.

D.   OBJECTIVES

This thesis applies Modeling and Simulation (M&S) concepts with multiple complimentary NPS software projects to explore the tactical capabilities of the Seadiver UUV. The objective of this research is to create a simulation using Discrete Event Simulation (DES) methodology and NPS simulation software tools. The simulation is designed based on the projected physical characteristics of the Seadiver UUV and other moving entities such as surface ships and submarines in the context of UUV missions. Simulation runs will then be conducted to determine the probability of Seadiver conducting exemplar missions such as barrier search or mobile minefield. Validation of mission results will be accomplished through evaluation of Seadiver behavior against environmental forces in the Autonomous Unmanned Vehicle Workbench (AUVW).

E.   THESIS ORGANIZATION

Chapter II reviews background technologies and related work used during this research effort. For each section, a short description is provided to give the reader a baseline understanding of topics that are referenced throughout the remainder of the thesis. Chapter III discusses the type of simulation framework (DES) along with the software products and Application Programming Interfaces (API) used in construction and validation of the simulation. Chapter IV outlines in detail the simulation design process using DES methodology and implementing model behaviors. Chapter V outlines in detail the creation of the Seadiver and other mover entity models and mission simulations. Chapter VI discusses the tactical considerations encountered using UUVs in novel mission contexts. Chapter VII discusses the results and analysis of the simulation. Chapter VIII is the conclusion of the thesis along with recommendation for further research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND RELATED WORK

A. INTRODUCTION

This chapter provides a conceptual overview of the technologies and related work utilized for this research. The following sections are intended to provide the reader with a basic understanding of these resources and how they were utilized in this thesis. Most sections provide links to these resources when appropriate.

B. SEADIVER GLIDER

Seadiver is a prototype UUV (Figure 1) being jointly designed by The Institute of Engineering Science of Toulon (ISITV), France with support from NPS. In the past decade, there have been many UUVs constructed, but what makes Seadiver unique and interesting for certain military applications is its combination of long autonomous operation time, relative inexpensiveness, and variable payload at virtually no extra operational cost.
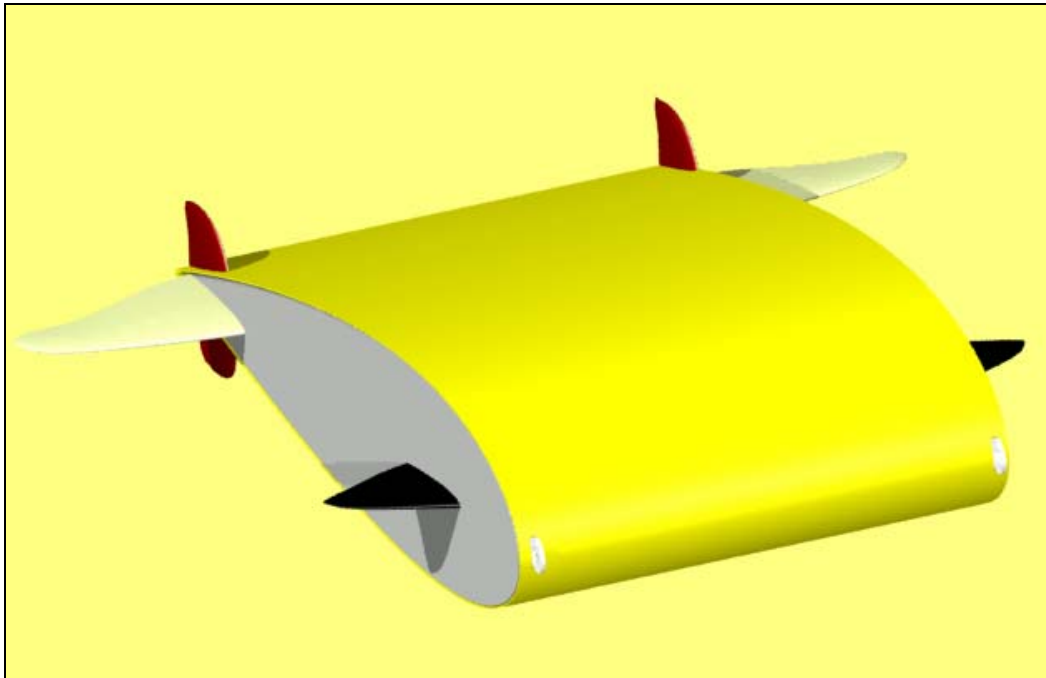
Figure 1.   Seadiver Glider 3D model. (from Dumonteil, Gassier, and Rebello 2006)

5

Seadiver is a type of UUV known as a glider. Gliders are unique in that they generate propulsion by managing vehicle buoyancy and therefore have no propellers or thrusters as shown in Figure 1. Propulsion is generated by hydrodynamic lift and drag. Seadiver moves from location to location by continuously adjusting buoyancy and angle of attack as shown in Figure 2. This has the effect of diving then ascending to generate speed over ground.
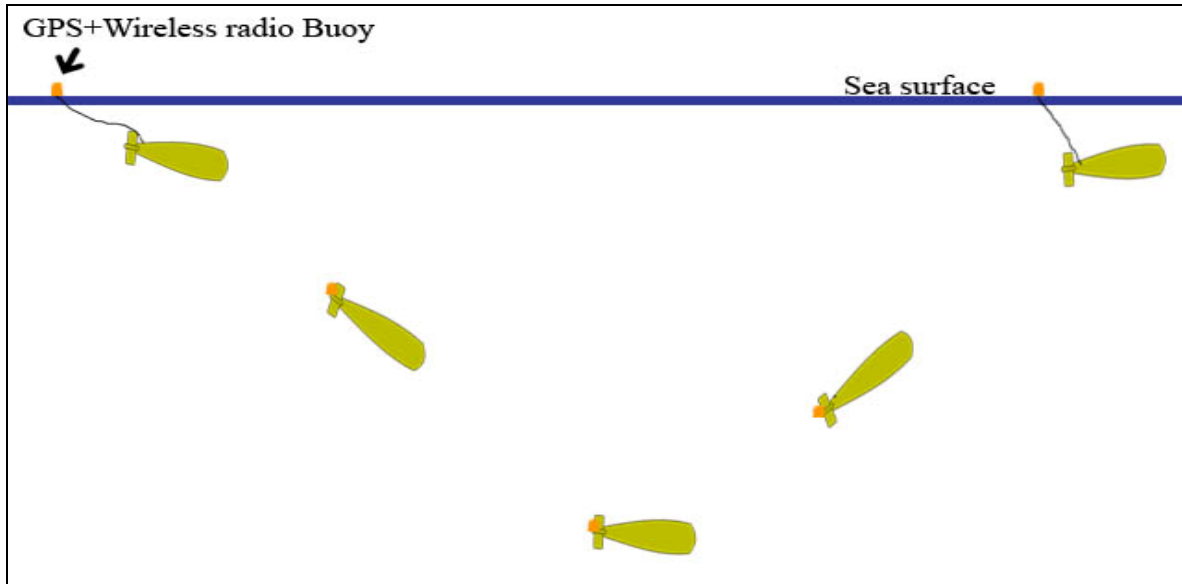
Figure 2.   Simplified plan of Seadiver's underwater behavior. (from Dumonteil, Gassier, and Rebello 2006)

Hydrodynamic properties are optimized by using an airfoil shape profile seen in Figure 3. This type of propulsion is very power efficient, but has the drawback of low speed across ground and maneuverability. Also, sensor position relative to the horizontal changes significantly from ascent to decent and must be accounted for in the design.
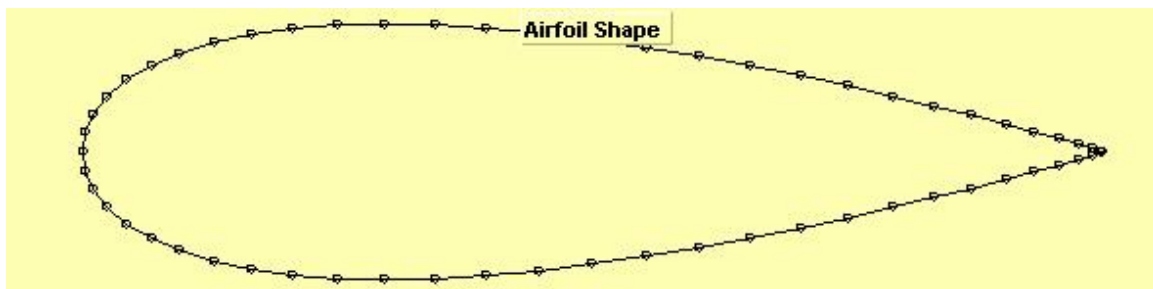
Figure 3.   Airfoil Shape (NACA0022). (from Dumonteil, Gassier, and Rebello 2006)

The main advantage of operating a glider is its ability to operate continuously for relatively long periods of time without recharging/refueling. This coupled with Seadiver's autonomy will allow it to remain on station operating independently for an estimated 30 days and travel approximately 700 NM depending upon payload power requirements.

Another major benefit of the Seadiver derived from its mode of propulsion and shape is that potential payload sizes are virtually limitless. The outer shape can be enlarged to accommodate many different size or use objects without having to redesign the entire UUV. This is because most of the interior is free-flood area where any inserted payload displaces water and therefore has a reduced effect on Seadiver's buoyancy and center of gravity.

C.   AUVW

The Autonomous Unmanned Vehicle Workbench (AUVW) is an open source software project designed and created at NPS that provides the ability to plan, rehearse, and replay missions for arbitrary unmanned vehicles (UV). It is designed to allow dissimilar vehicles to be evaluated on a common software platform which is normally difficult since most UVs use proprietary vehicle specific data formats and mission planning systems (Davis and Brutzman, 2005). Figure 4 depicts 2D mission planning and Figure 5 depicts 3D mission playback in AUVW. As shown, AUVW provides a tightly coupled 2D/3D interface that simplifies UV testing and operation.
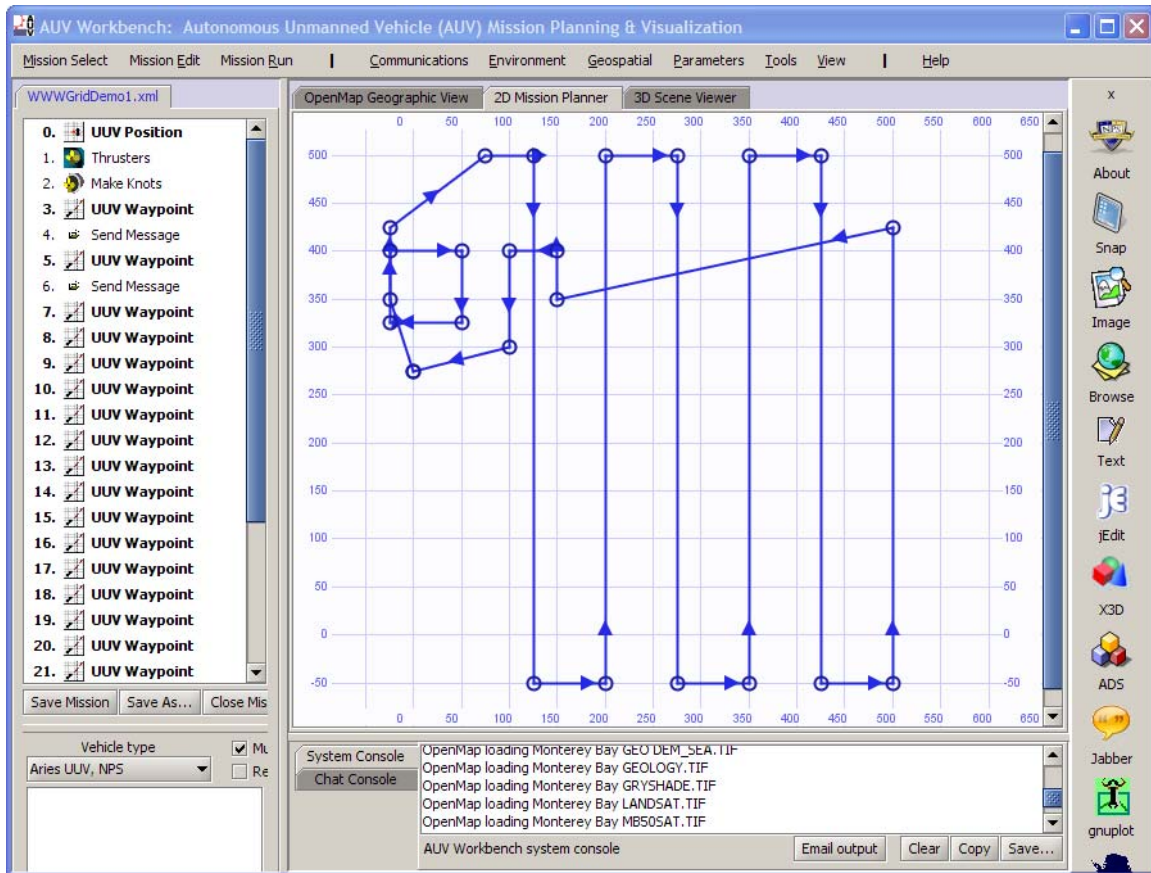
Figure 4. 2D mission planning in AUVW.
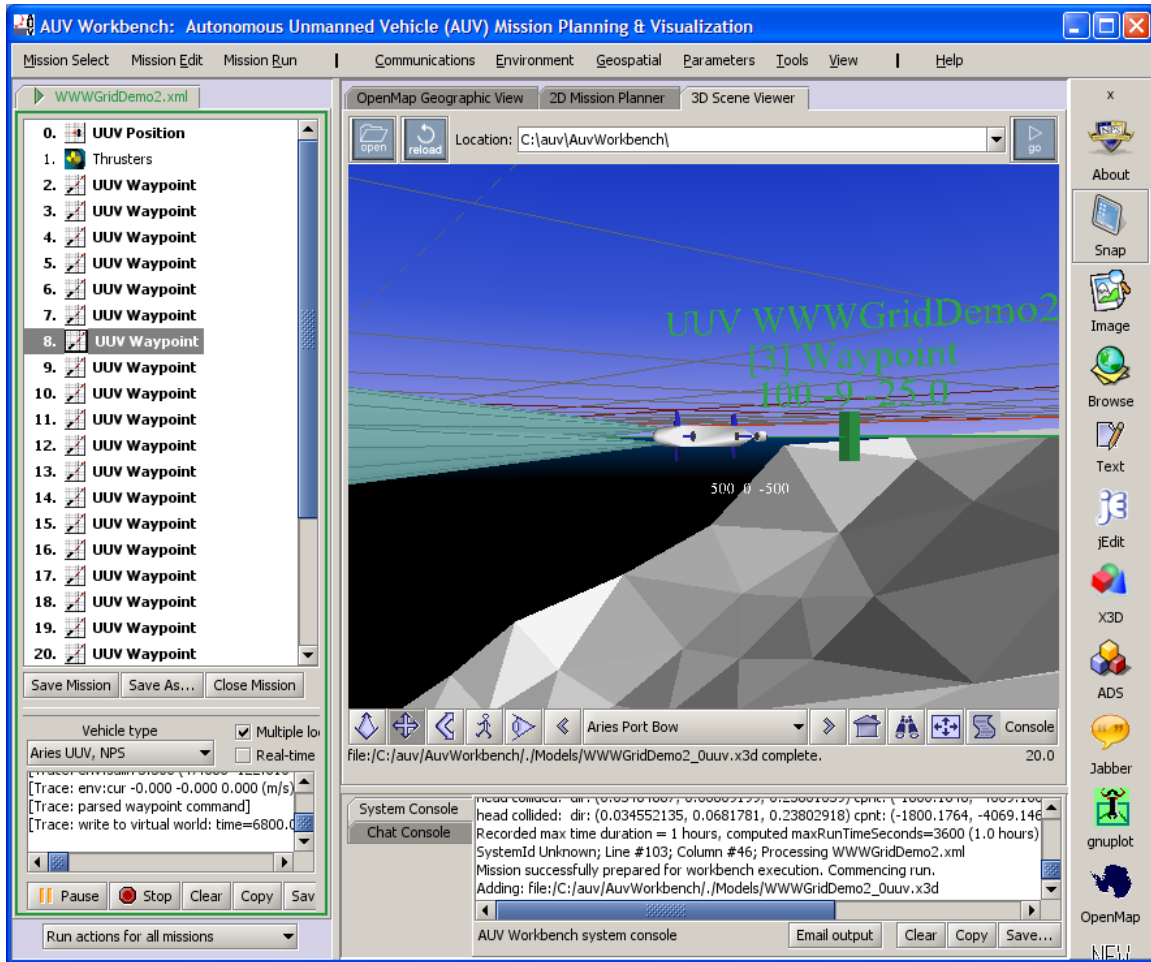
Figure 5.   3D mission playback in AUVW.

The AUVW is used in this thesis to provide validation for the output missions (discussed in detail later) of the Seadiver simulation. This is accomplished in two ways. First, output missions are visually verified to be true in the context of operating area dimensions, vehicle area dimensions, vehicle waypoint placement, and proper search pattern implementation as defined in the simulation initial conditions (event graph parameters). Second, AUVW has the ability to simulate the environment and physical characteristics of the vehicle in six degrees of freedom. This allows validation of entity movement and physical performance in a virtual environment based on the real world physical conditions and constraints. Figure 6 depicts the AUVW dataflow model.
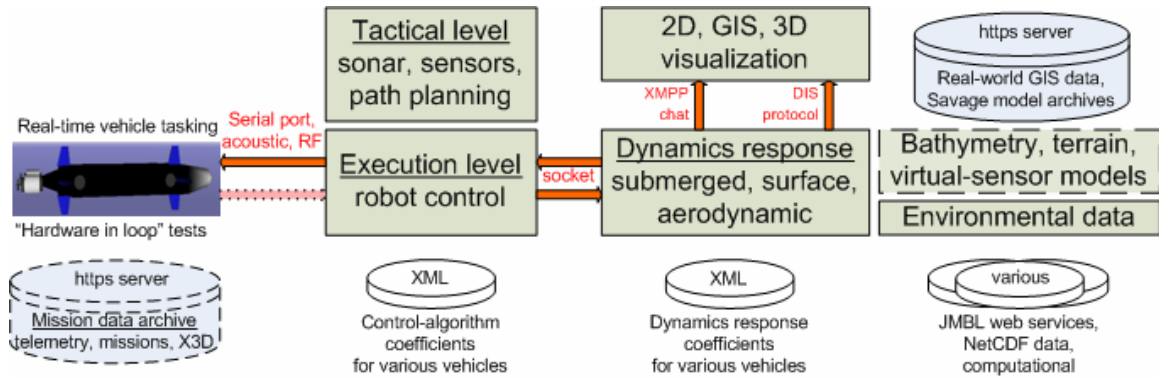
Figure 6.   AUVW dataflow.

The AUVW also contains an extensive help system which can be used as a resource for thesis students and other users. The help system is accessed from the menu bar and contains tutorials and menu descriptions for users new to the workbench along with all theses and dissertations that have utilized it previously. The AUVW help system is a valuable resource for operation and research.

D.   PROGRAMMING CONSTRUCTS

1.   JAVA

Java is an object-orientated programming language developed by Sun Microsystems. Java was designed to be platform independent so a developer could write a program once and run it on any arbitrary set of computer hardware. Java is used extensively at NPS for that reason and because most Java development tools are free. Java is primarily used for Modeling and Simulation because of its platform independent design, its multi-threaded capability, and the multitude of available related open-source code such as JSIM, X3D, etc.

2.   JAXB

JAXB is an open-source API created by SUN Microsystems. It provides a convenient way to bind XML schemas to java source-code representations. JAXB makes it easy for developers to incorporate XML data and processing into applications. As part of this process, XML documents are either marshaled to java classes or unmarshaled into a JDOM tree for use by the program.

VISKIT stores event graph models as XML documents. Figure 7 depicts a simple event graph, the Arrival Process, and Figure 8 is the XML representation of it in Viskit. JAXB enables the XML structures used by Viskit to store event graphs to be transformed into executable java source (Figure 9) that can then be utilized by Simkit and Diskit. This allows developers to create DES models quickly using only standard event graph notation and methodology without having to master the Java programming language.

Figure 7.   Arrival Process event graph.
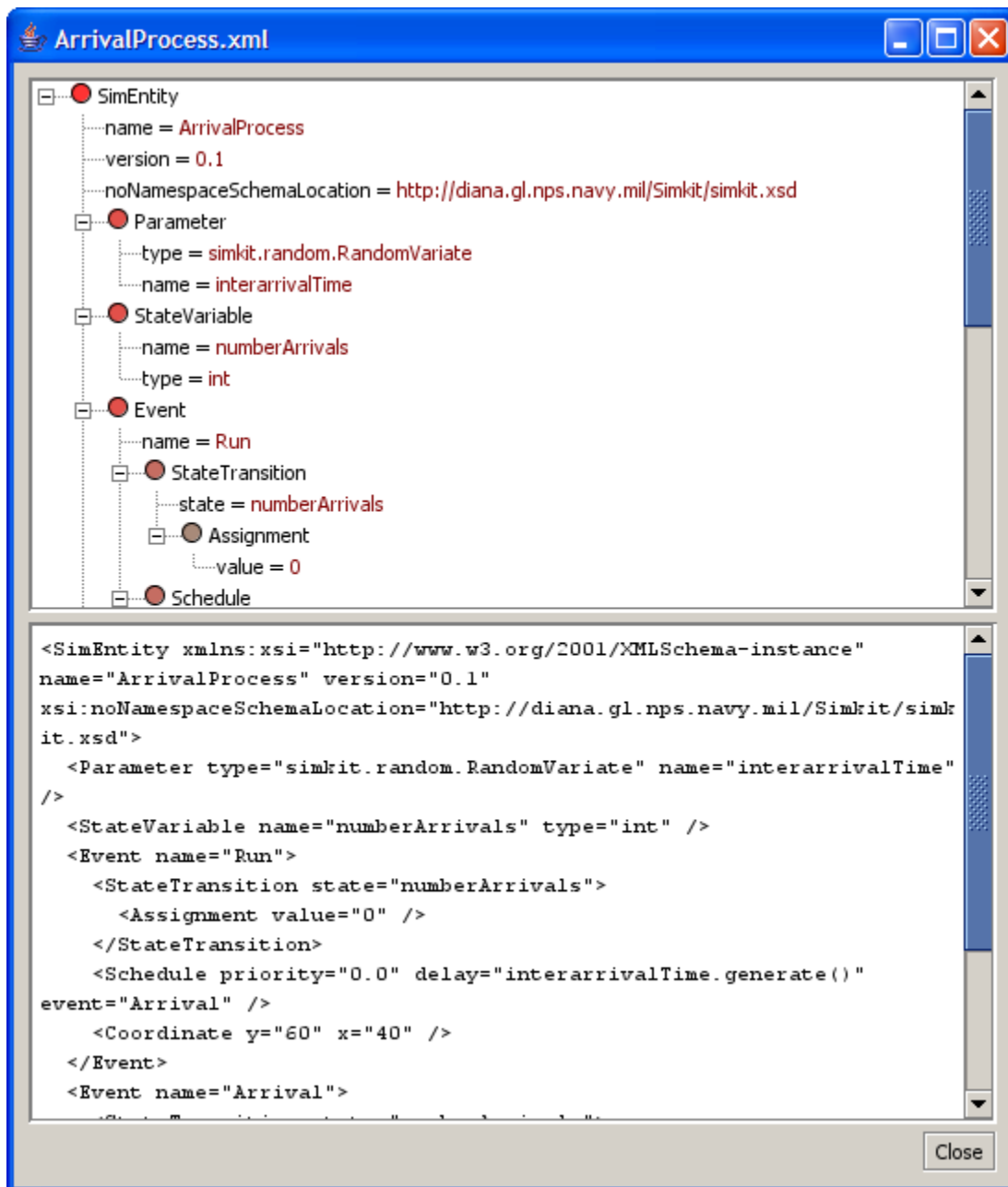
Figure 8.   Viskit event graph XML structure. The top pane is a more human-
readable tree-view, while the bottom pane shows source XML file.

```
1: package examples;
2:
3: import simkit.*;
4: import simkit.random.*;
5: import java.util.*;
6:
7: public class ArrivalProcess extends SimEntityBase {
8:
9:     private simkit.random.RandomVariate interarrivalTime;
10:
11:    protected int numberArrivals;
12:
13:    /** Creates a new instance of ArrivalProcess */
14:
15:    public ArrivalProcess(simkit.random.RandomVariate interarrivalTime) {
16:
17:        setInterarrivalTime(interarrivalTime);
18:    }
19:
20:    /** Set initial values of all state variables */
21:    public void reset() {
22:
23:        super.reset();
24:
25:        /** StateTransitions for the Run Event */
26:
27:        numberArrivals = 0;
28:    }
29:
30:    public void doRun() {
31:        firePropertyChange("numberArrivals",numberArrivals);
32:        waitDelay("Arrival",interarrivalTime.generate(),Priority.DEFAULT);
33:
34:         }
```

Figure 9.   Automatic source-code generation in Viskit from source XML event graph in Figure 6 above.

3.   Document Object Model (DOM)

The Document Object Model (DOM) is a platform and language-neutral interface and World Wide Web Consortium specification that allows programs and scripts to dynamically access and update the content, structure and style of documents.   Sun Microsystems has implemented the DOM interface a component API of JAXP in the org.w3c.dom Package. It allows programmers to create, modify, access, and write XML documents using the Java programming language. Additional information is available at Sun's   website   at   http://java.sun.com/j2se/1.4.2/docs/api/org/w3c/dom/package-

13

summary.html (accessed March 2007). The DOM is used in this thesis to create AUVW mission files in Autonomous Vehicle Command Language (AVCL) format for validation of the Seadiver simulation.

4. Extensible Markup Language (XML)

Extensible Markup Language (XML) is a general purpose, text based markup language developed by the World Wide Web Consortium (W3C) as a subset of Standard Generalized Markup Language (SGML). Like all markup languages, it was created as a protocol for structuring data. It is not a programming language but it makes it easy for a computer to generate data, read data, and ensure that the data structure is unambiguous. XML is easy to create and process and designed to be platform independent and shared across the internet. Other characteristics of XML include human readability, extensible, verbose, modular, and license free. More information can be found at www.w3.org/XML/1999/XML-in-10-points (accessed March 2007).

XML is used in Viskit as the format for saving Event Graphs and assemblies, and in the AUVW to store UUV mission files. These mission files are in Autonomous Vehicle Control Language (AVCL) format which is valid XML syntax. Figure 10 is an example XML representation of a notional restaurant price list.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- edited with XMLSpy v2006 sp2 U (http://www.altova.com) by John (Naval Postgraduate School) -->
<!-- Edited with XML Spy v2007 (http://www.altova.com) -->
<breakfast_menu>
   <food>
      <name>Belgian Waffles</name>
      <price>$5.95</price>
      <description>two of our famous Belgian Waffles with plenty of real maple syrup</description>
      <calories>650</calories>
   </food>
   <food>
      <name>French Toast</name>
      <price>$4.50</price>
      <description>thick slices made from our homemade sourdough bread</description>
      <calories>600</calories>
   </food>
</breakfast_menu>
```

Figure 10.   Simple XML file.

E.   **SUMMARY**

This chapter has provided the reader with an overview of the technologies and related work utilized for this research. Section B detailed the novel glider UUV that provides the inspiration for this thesis. Section C describes the simulation and virtual reality program (AUVW) which is used for mission validation. Finally, Section D describes the main programming constructs leveraged by the software APIs and programs employed in this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SIMULATION AND PROGRAMMING CONSIDERATIONS

A. INTRODUCTION

This chapter describes in detail the simulation framework and software programs and APIs that leverage this framework. Section B defines Discrete-Event Simulation (DES) as the theoretical framework in which this thesis creates a model of the Seadiver UUV and other entities. Section C and D describe the Simkit and Diskit APIs and how they are leveraged in this research. Finally, Section E explains the functionality of Viskit as the main simulation environment of this thesis.

B. DISCRETE-EVENT SIMULATION (DES)

1. Modeling Characteristics

Models are created to study complex dynamic systems and examine their performance, reliability, or other properties to improve either their initial design or operation. Simulation is the means of executing these models to mimic the behavior of actual systems. Simulations employ many repetitive runs to obtain relevant statistical output for insight into the actual operation of the modeled system without real-world testing that is often impractical and costly. In general, if a model uses an equation to define a characteristic, then a simulation is the behavior or trajectory of that function over time.

There are two defining characteristics when creating a model. The first, fidelity, measures the level to which the model reflects the characteristics of the real system like how similar it is in shape/dimensions, physical characteristics/constraints, or performance. The second, abstractness, measures the lack of level of detail. This is required not simply because it is impossible to capture every detail of a real-world system which may or may not be known, but also because it allows for generality. Generality is beneficial since it possibly allows for the design and analysis of multiple models simply by changing parameters. (Miller 2007)

The ideal model has high both abstractness and high fidelity. Unfortunately this is impossible and therefore a compromise must be struck that usually depends upon the needs of the modeler. This thesis is highly abstract because the modeled system is still in the design phase and not much is yet known about its real-world characteristics.

### 2.	Simulation Approaches for Handling Time

There are two broad types of simulation modeling primarily characterized by how they handle the passage of time. The first is Continuous Systems Simulation (CSS). CSS is creating a model that can be represented by differential or difference equations. In essence, it breaks the time domain into quantized chunks of small (usually the same) size. This approach is used by the AUVW which serves as a validation tool for this thesis. The second is DES and is also the focus of this thesis. It differs from CSS because it divides the time domain by events. According to Professor Arnold Buss of NPS, DES has three main world views; Event-Scheduling, Process-Interaction, and Activity Scanning. The Event-Scheduling approach is based on the use of event lists to organize future events. This is the world-view utilized in Simkit, and therefore is utilized in this thesis.

### 3.	Methodology

Events are actions defined by the modeler to represent basic functionality of the simulation. They represent changes in state that typically takes some amount of time to occur such as an object arriving to the queue or a server completing a job. The event list is simply a container that holds the list of events that are scheduled to happen and the time at which they will happen. Buss describes the event list as:

> The Event List amounts to a "to do" list for the simulated world. At any simulated time epoch it is simply a list of what is scheduled to occur and when. Each item of the list corresponds to an event that contains information about which event is to occur and when it is to occur. (Buss 2000)

The scheduling and manipulation of the event list is the engine driving a DES. Every action that comprises the model will be scheduled on the event list. Time advances only in intervals defined by the time difference between the current event time and the event on the event list with the smallest time duration. This process continues throughout

the duration of the simulation, that is each event is drawn off the event list one at a time ordered by the time the event is scheduled to occur, until the event list is empty or an event is scheduled that explicitly stops the simulation. Note that it is possible for two events to be scheduled for exactly the same time and therefore it is necessary to implement an order of precedence procedure in the event list.

4.   Notation

An event graph is a structured, formal representation of a DES model. Event graph notation was defined in work by Schruben in 1992. This notation is minimalist in that it uses only those entities that are required, but in doing so add a level of abstractness not seen in other DES world views such as Process-Interaction (Buss 2001). The advantages of adhering to this notation are that virtually any model can be constructed with it and the modeler can spend more time on model creation vice paradigm constructs. Using the following notation conventions, the modeler can graphically depict all logic and behavior contained in the model.

The most basic event graph is depicted in Figure 11. The two objects fundamental to every event graph are the event nodes represented by circles with labels and scheduling edges represented by directed lines or arcs. In Figure 11, event node A is an event that can appear on the event list. The directed arc above event A is the scheduling edge (in this case a self-scheduling edge since it re-schedules itself). In essence, this event graph depicts an event A that will continuously re-schedule itself unless interrupted by an outside event. Additionally, this event graph provides no method to begin the initial event A and therefore it must be initialized by a foreign event either programmatically or through a listener pattern (described later). Normally every event graph contains a Run event from which other events are propagated and to reset all state variables when a simulation run is completed.
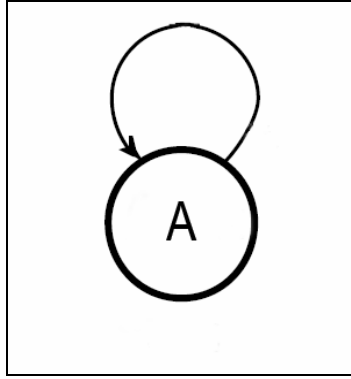
Figure 11.   Simplest Event Graph.

Two optional components of scheduling edges that greatly extend the functionality of event graphs are <u>edge conditions</u> and <u>time delays</u>. Represented in Figure 12, edge conditions are conditional expressions defined by the modeler that prevents the edge from being invoked until said condition is true. Edge conditions are represented by logic functions above the wavy line in the middle of the scheduling edge. Time delays, represented in Figure 12 by (t) located at the start of the scheduling edge, control exactly when from execution of event A that event B is to be scheduled. Therefore, this event graph depicts that once event A is scheduled, event B will be scheduled (t) amount of time later if expression (*i*) is true.
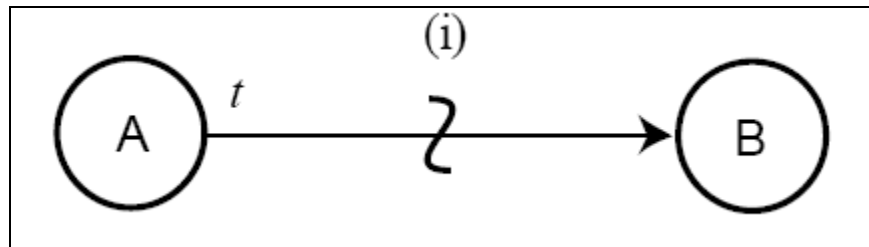
Figure 12.   Figure of next event graph 'A' and 'B' connected by a scheduling edge with time delay (t) and conditional expression (*i*).

Two final and necessary components of complex event graphs are parameters and state variables. Parameters area variables defined at the start of each simulation run and represent constructs such as total number of servers or number of targets to be created. State variables on the other hand are variables designed to change throughout the simulation run. As the name suggests, state variables are updated to reflect the changed

20

state of the model such as the number of people in the queue at a particular time or if an UUV is currently surfaced or submerged. Using this notation, it is possible to construct models of limitless complexity. Figure 13 depicts a more complex model of a transfer line process where a component is passed from one server to another and is finished only when processed by all servers. In this model, Q and S represent state variables and (*i*) represent a passed parameter.
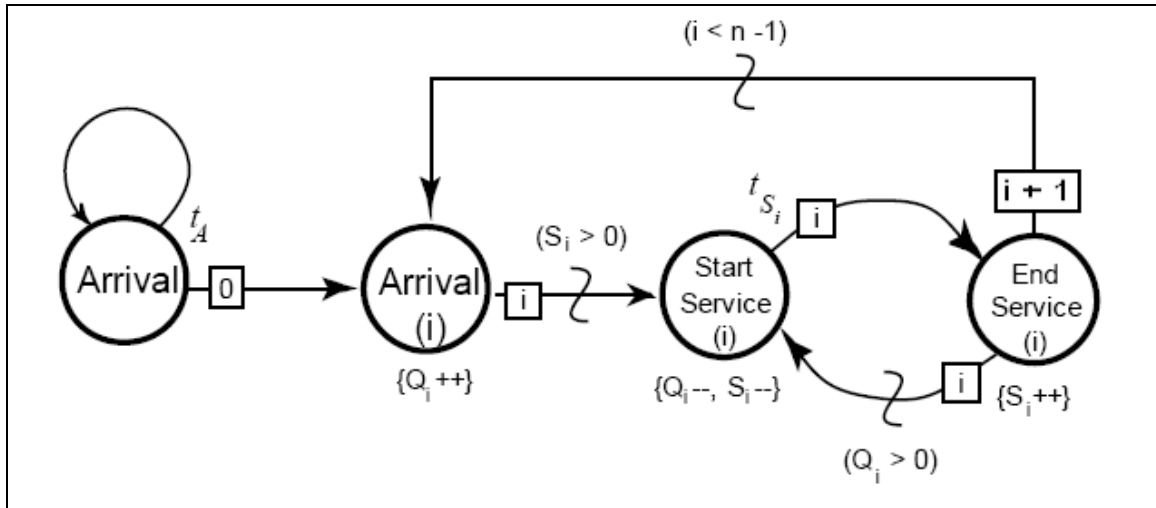
Figure 13.   Complex Event Graph of a Transfer Line Process (from Buss 2001)

C.   SIMKIT

Simkit is an open source Java API written by Professor Arnold Buss of Naval Postgraduate School. It was designed to enable the creation of DES using event graph methodology. In short, a simulation can be created programmatically using Simkit because it provides the base framework for controlling the simulation, namely control and maintenance of the Event List. This frees the modeler to work directly on implementing the conceptual event graph model. Simkit also provides other helper classes necessary to create simulations. These include random variate generators that produce random numbers in the required distributions, and classes that facilitate movement and detection among others. Simkit is the foundation upon which Diskit and Viskit are built. It is possible to create complex DES using Simkit and the following website lists the NPS Master's Thesis work that has been completed using Simkit (http://diana.nps.edu/~ahbuss/#Students accessed on March 2007).

D.   DISKIT

Diskit is another open source Java API that extends the functionality of Simkit. It was created for two primary reasons. First, there was a need to extend the movement and detection capabilities of Simkit to 3D. This is because 2D usually doesn't provide the level of fidelity required for a model that simulates movement. Secondly, Diskit provides classes that implement the Distributed Interactive Simulation (DIS) protocol. DIS allows for transmitting the state of a simulation over a network. DIS coupled with the extension to a 3D environment enable the visualization of the simulation as a 3D virtual environment.

E.   VISKIT

Viskit is an open source program in development at NPS written in the Java programming language. Viskit was created to provide a graphical user interface (GUI) for creating simulations using Simkit.   Typically, creating complex simulations is programmatically intensive. A modeler usually needs an extensive knowledge of a programming language and the associated APIs that enable the simulation. This is no different for Simkit and Diskit, and is exactly why Viskit was developed. By reducing the amount of programming expertise required, Viskit has made simulation more accessible to non-programmers.

Viskit uses a tabbed window with four tabs. The first provides a visual interface that allows for easily creating, modifying, and saving event graphs called the event graph editor. Figure 14 provides an example of a simple event graph in Viskit's event graph editor. It demonstrates that event graphs produced in Viskit faithfully adhere to the event graph methodology presented earlier in Chapter II. This ensures that if event graph methodology is understood, Viskit can represent it and others who have no familiarity with Simkit or Diskit can understand it. Additionally, because the source code is automatically produced by Viskit, it allows for more complex event graphs to be created without being increasingly encumbered with programming complexity that might quickly become unmanageable.
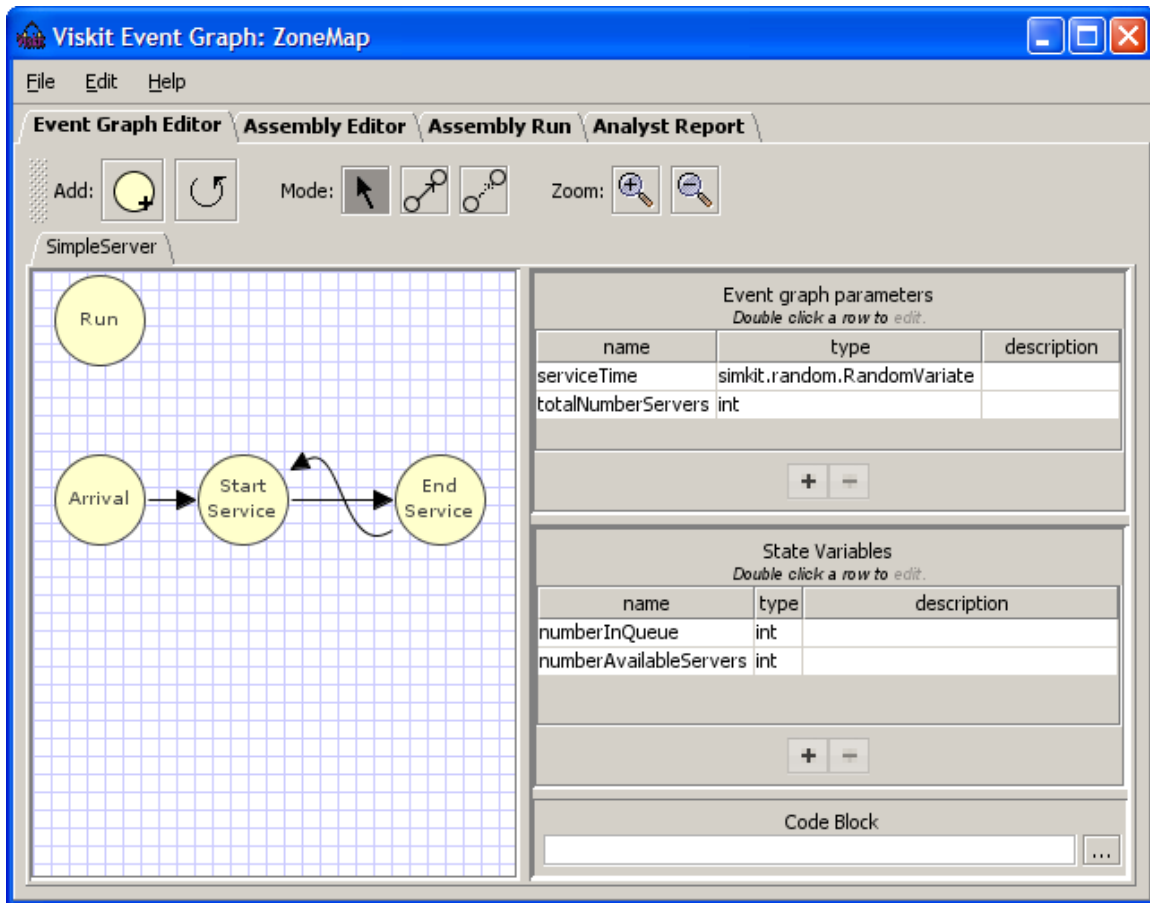
Figure 14.   Viskit's Event Graph Editor Panel depicting a basic example.

Creation of the event graph representation of a model alone does not create a run able discrete-event simulation. Viskit provides a means to create, modify, and save simulations using event graphs in a panel called the Assembly editor located in the second tab. Figure 15 depicts a simple simulation setup in the assembly editor. Event graphs that were created in the event graph editor (or any event graph created that extends Simkit's SimEntityBase) show up on the left panel and are drag and dropped to the right workspace. They are then connected using listener patterns (discussed later). Finally, statistics-counting objects are listed in the lower left panel and drag and dropped to the workspace on the right as required where they are connected to the event graphs with PropertyChangeListeners (discussed later). This will produce applicable and repeatable statistics as required of the simulation.
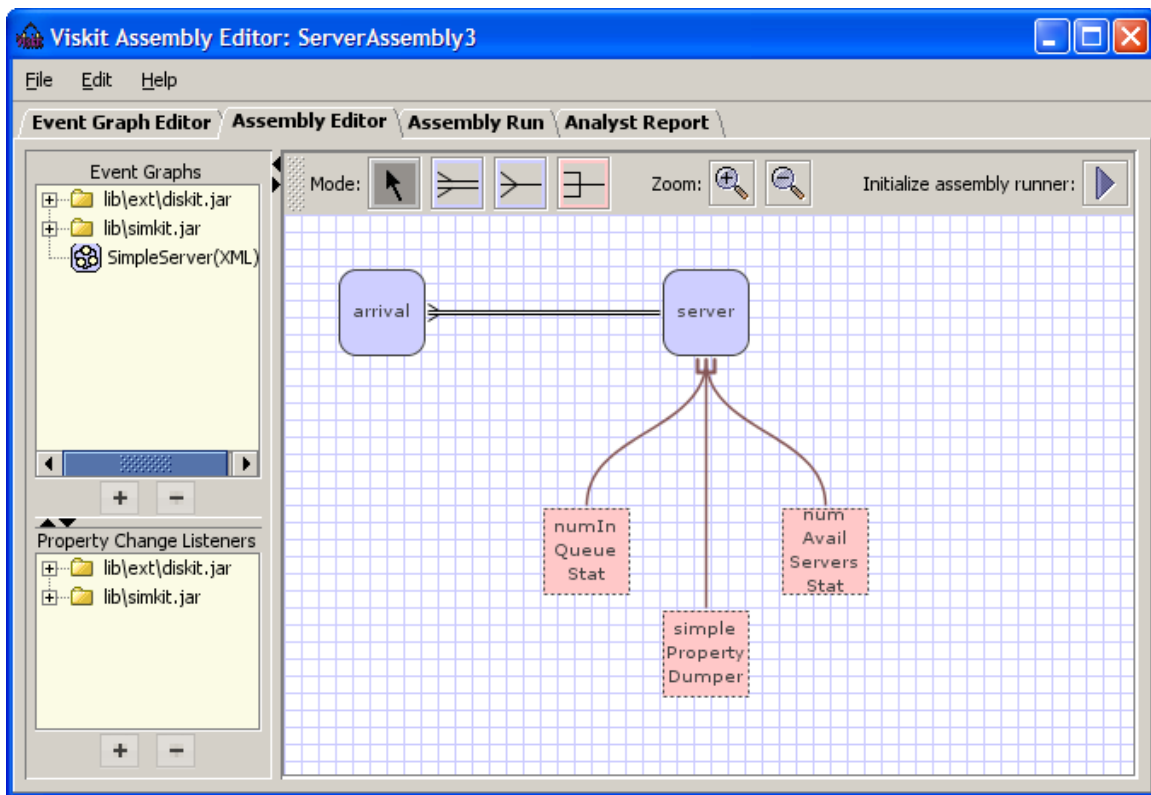
Figure 15.   Viskit's Assembly Editor Panel depicting a basic example.

The Assembly Run is the third tab and provides a location to run the simulation. Figure 16 depicts the Assembly Run panel after a run of an exemplar simulation. On the left are controls to modify run parameters of the simulation such as length of time to run and how many times to run the simulation. The top right of the Assembly Run panel contains the text output of the simulation that can be inspected after each run. The bottom right of the panel provides an error report generated during the run.
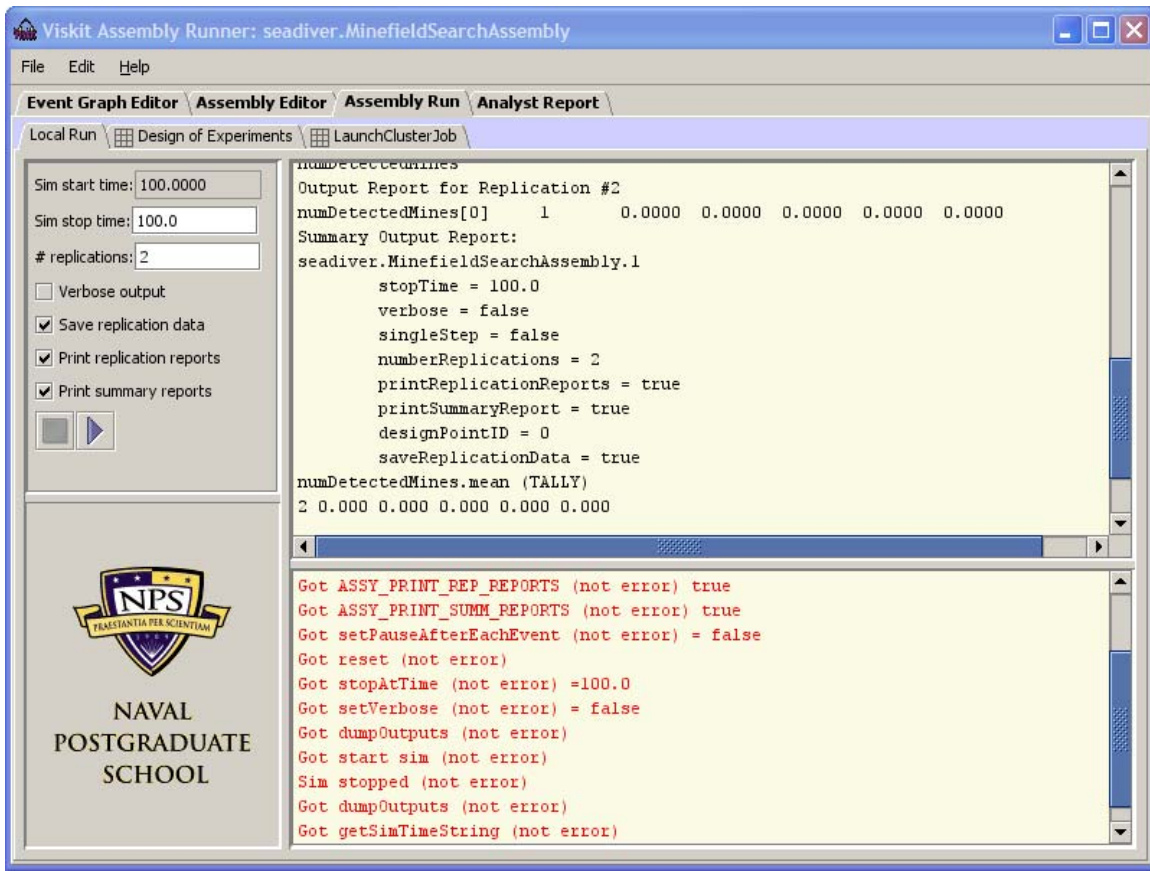
Figure 16.   Viskit's Assembly Run Panel.

The last tab is the Analyst Report Editor. It provides an interface to enter detailed information about the model and simulation runs that can be saved in XML format. This combined with the simulation output generate a standardized report of the simulation quickly and easily. The Analyst Report panel is itself composed of a number of tabs each detailing with an aspect of the simulation and report. Figure 17 depicts the Analyst Report Editor panel at the beginning tab called 'Heading' that collects information on the simulation such as title and author. In general, each section of the analyst report has two parts. First the analyst records what behaviors and results are expected to be produced by the model. Second, the analyst assesses what was actually produced. The tool is set up to encourage incremental analysis and testing, saving intermediate analysis each time. Once each required tab is documented, a complete analyst report is generated in HTML format to match. Appendix A is an example analyst report of a Seadiver mission.
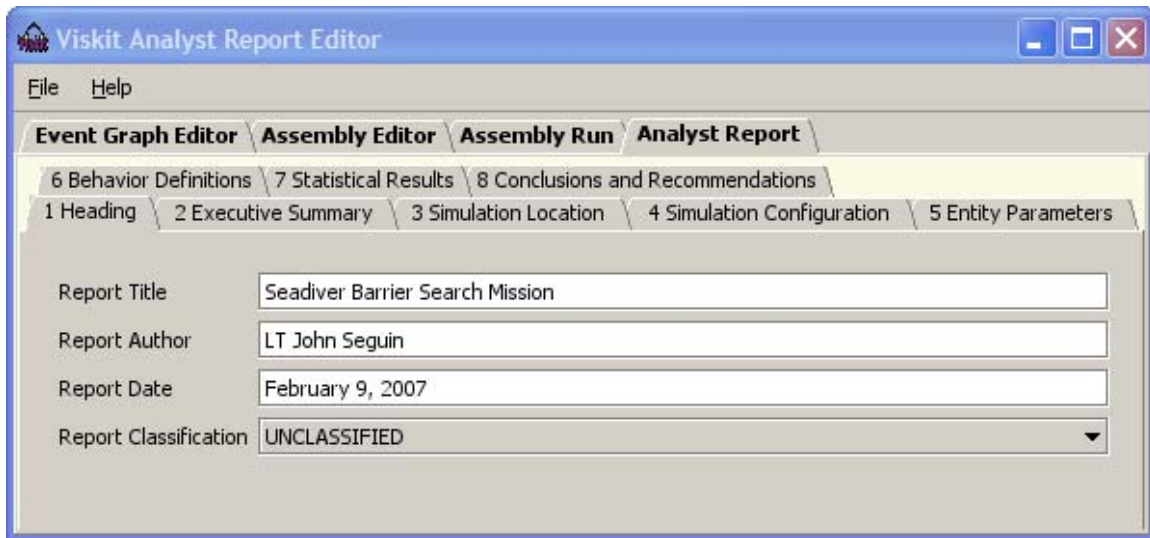
Figure 17.　　Viskit's Analyst Report Editor panel excerpt.

F.　　SUMMARY

This chapter detailed the simulation framework along with the software programs and APIs leveraged by this framework. This thesis utilizes DES methodology and notation for creation of the Seadiver model. This is accomplished by employing Viskit which is a program that provides a GUI for the creation of DES event graphs and simulations. Viskit enables efficient creation of DES models by leveraging the Simkit and Diskit APIs.

IV. DES AUTHORING – CREATING A SIMULATION WITH VISKIT

A. INTRODUCTION

This chapter describes in detail how to create a DES in Viskit. Every modern programming language enables a construct called inheritance which facilitates code reuse while decoupling form from function. Inheritance and how it is leveraged in this thesis is the subject of Section B. Section C details event graph authoring in Viskit. Event graphs are the main logical constructs in DES and define entity behavior. Section D explains how event graphs are integrated into an assembly in Viskit to produce a simulation. Finally, Section E illuminates how movement and detection functionality is implemented in Simkit/Diskit.

B. SIMKIT/DISKIT API LIBRARY INHERITANCE STRUCTURE AND USE IN VISKIT

Computer programs are complex constructs that if coded in a single container would extend many lines and pages. Java and indeed most programming languages provide mechanisms to organize and reuse code as much as possible. Viskit simulation architecture utilizes all those inherent to Java, but one is of particular interest in Viskit called inheritance. Inheritance allows for many implementing objects to contain all the inherited characteristics of the superclass while allowing for the addition of a new functionality in the current class. Additionally, the current class could then be used as the superclass for another class, etc, etc. In effect, inheritance provides the ability to create multiple entities that are primarily equivalent, yet have unique functionality.

The concept of inheritance is especially important to entity creation in Viskit. As entities become more complex thru the process of implementation of features, the event graphs can become overwhelmingly complex as shown in Figure 18. While Figure 18 is a fully functioning Viskit entity, it is clearly difficult to decipher, modify, or test for desired behavior. Figure 18 is an attempt to implement tactical behavior in Viskit without inheritance.
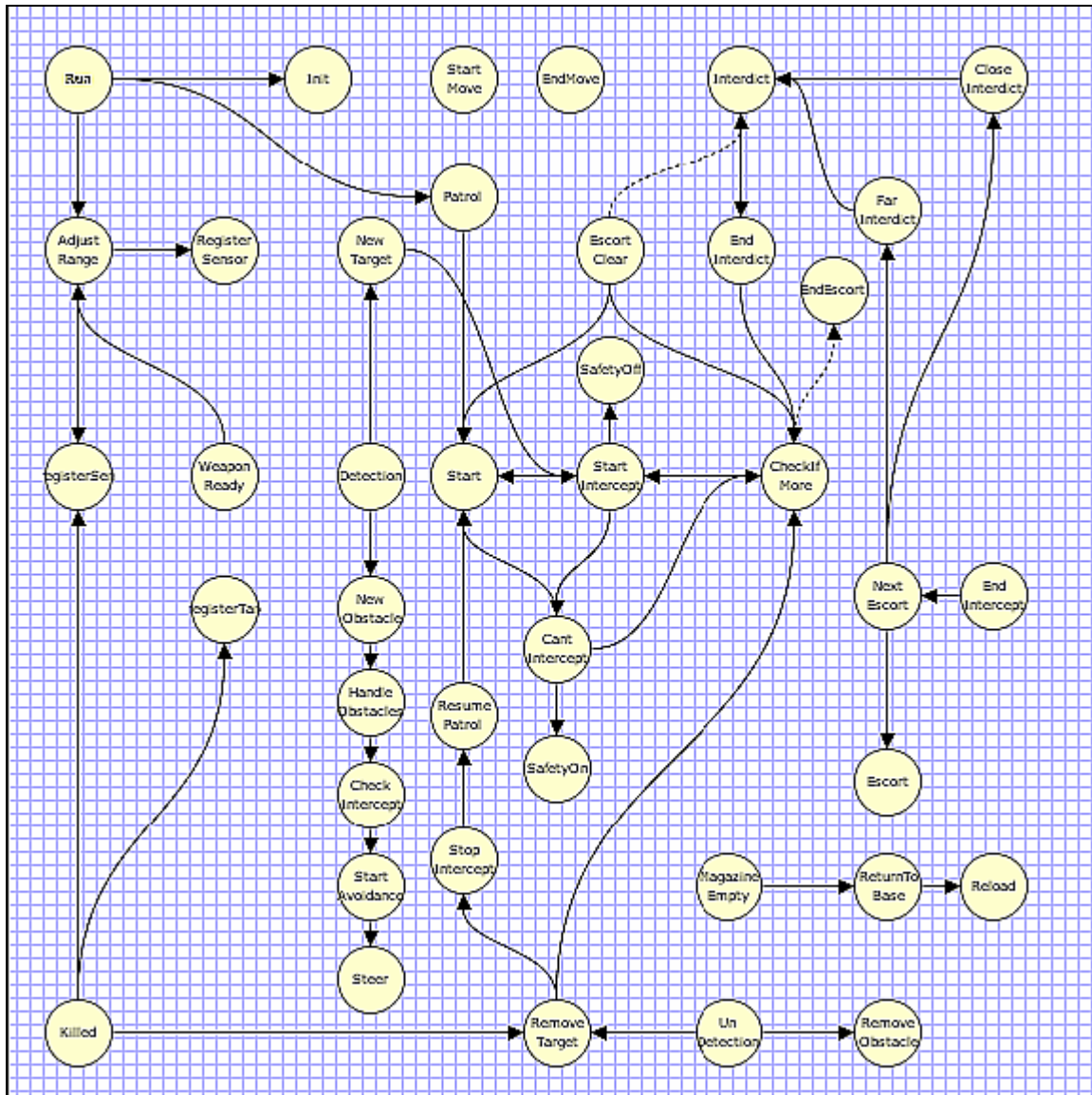
Figure 18.   Complex event graph without inheritance (from Sullivan 2006).

Viskit allows for the use of inheritance and it has been demonstrated that its judicious use is essential to producing a maintainable model and is a 'best practice' and simply a good design pattern that should be followed. Viskit provides this ability thru the event graph settings dialog box that allows for specifying which class to extend. It is important to note that to be used in Viskit, at some point one of the super classes must be SimEntityBase. The benefits to adhering to this best practice include event graphs that are more readable, focused on implementing only what is different from the superclass, and easier to debug.

1. SimEntityBase

SimEntityBase is the fundamental component of Simkit simulations. Recall from Chapter 2 that there are just two constructs of event graphs: the event and the scheduling edge. SimEntityBase is the class that controls interactions with the event list. Each event on an event graph is placed or removed from the event list according to its scheduling edge. Every event graph in a simulation or one of its super classes must inherit from SimEntityBase at some point otherwise it cannot interact with the event list. A more detailed discussion of SimEntityBase is provided in (Buss 2002).

2. Mover3D

Mover3D is a Java interface that ensures implementing classes meet the minimum requirements for a 3D mover in Diskit. It is essential that all movers in Simkit requiring interactions such as detection and its inverse, undetection, implement Mover3D. This is because of how the sensor classes are constructed since they fire 'doDetection' and 'doUndetection' events as specified by Mover3D. By convention, implementing classes are named with Mover3D appended such as DISMover3D.

3. DISMover3D

DISMover3D implements the Mover3D interface and extends SimEntityBase, thus it provides the minimum constructs for a Simkit simulation as well as ensuring it will interact properly with the sensor library. Additionally, it provides all the functionality required for a moving entity along with exposing that entity to the DIS protocol. Essentially, DISMover3D provides all the functionality required of a simple 3D mover that can detect other object and output its state as DIS packets across a network. The Seadiver and Target event graphs in this simulation use DISMover3D as its moving entities. Figure 19 depicts the event graph representation of DISMover3D.
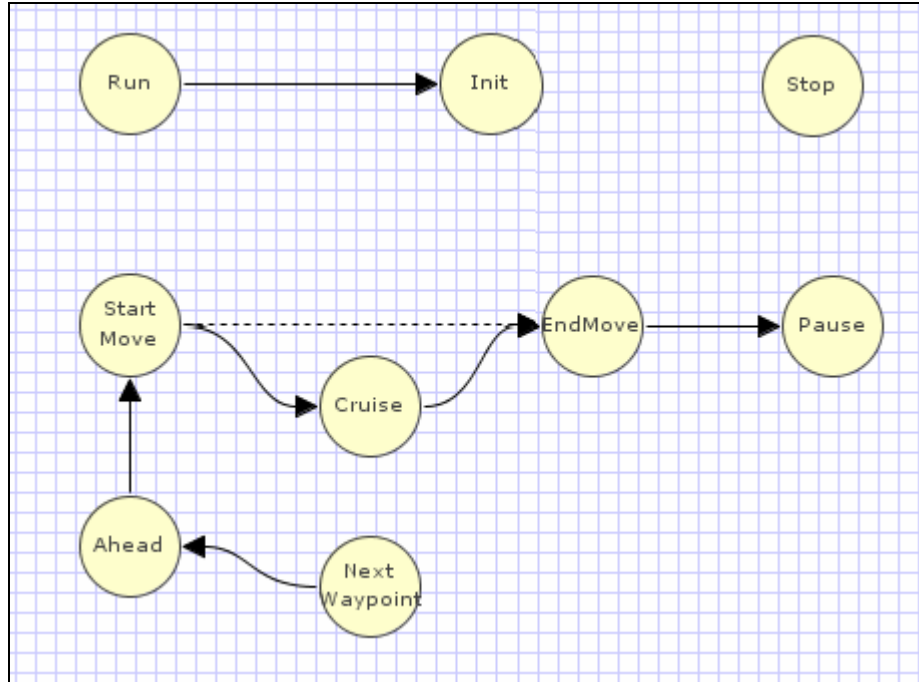
Figure 19.   DISMover3D Event Graph (from Sullivan 2006).

4.   Seadiver Model Inheritance Structure

The Seadiver simulation implements inheritance to the fullest extent possible. Figure 20 is a diagram depicting the simulation inheritance structure. Notice that all entities are at some point descendants of SimEntityBase. This is a requirement of Viskit which enables all descendants to be observable, selectable, and able to be integrated into assemblies. Along with extending SimEntityBase, all movers and sensors implement the Mover3D interface represented by the dashed line. This is required to enable movement and detection functionality through Diskit. Similarly, if only 2D movement and detection is required, Simkit has a corresponding Mover interface that must be implemented.
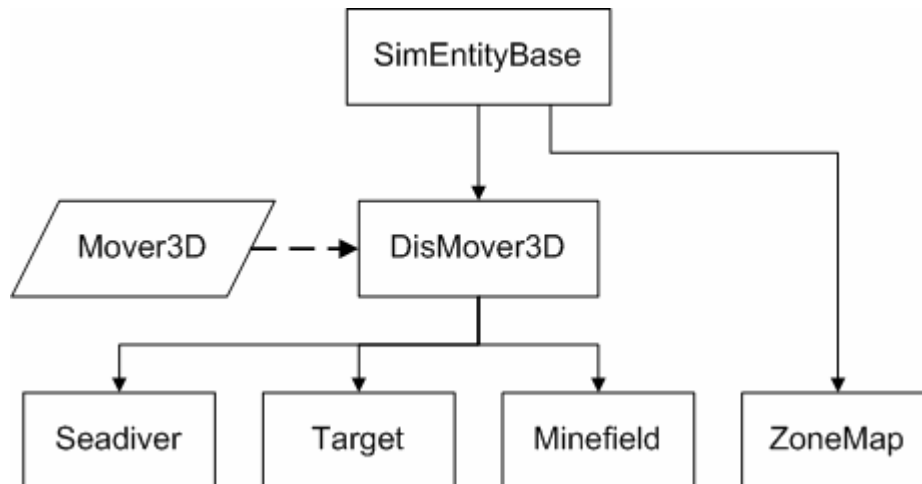
Figure 20.   Diagram of Seadiver inheritance structure.

C.   EVENT GRAPH EDITOR – CREATING A MODEL

Event graphs define a DES and control the behavior of the entities and their relationships to other entities. Producing a productive simulation requires creating event graphs that encompass behaviors of sufficient fidelity while maintaining some requisite amount of generality.

This discussion demonstrates how the event graphs in the Seadiver simulation were created in Viskit. It is not a full tutorial for Viskit, but does show how event-graph methodology was used to create the Seadiver event graph for the model in Viskit.

1.   Event Graph Parameters

In Simkit DES methodology, event-graph parameters are variables that are set at simulation run time and do not change during the run. The exact value of parameters must be entered in the Assembly Panel prior to the start of the simulation or an error will occur. Parameters also represent performance characteristics of a model such as sensor range or maximum speed. These must be available as changeable parameters to maintain a sufficient level of abstractness to allow for multiple simulation runs without modifying hard-coded values.

31

Viskit provides for view and modification of event-graph parameters in the Event Graph Editor panel. Figure 21 depicts the Seadiver event graph parameter list as displayed in Viskit. The plus and minus buttons at the bottom allow for addition or removal of parameters. To modify an existing parameter, simply double click the parameter line. These parameters will be discussed fully in a later section.

name	type	description
maximumSpeed	double	The maximum speed for this entity.
firstMoverID	int	The unique DIS entity ID number for first entity.
numberDivers	int	Total number of divers to be created.
sensorRange	double	Range of sensor (sonar,MAD,etc).
fixDelayTime	diskit.random.RandomVariateInstantiator	Random variate for duration in minutes to conduct a gps fix.
inputFileTemplate	String	Name and Path of file to parse for use as a template for generation of AUVW missions.
outputFileName	String	Name and path (without extention) that will be used to create and write seadiver AUVW mission files.
operatingDepth	double	Nominal operating depth of the seadiver during mission runs.

Event graph parameters
Double click a row to edit.

Figure 21.   Seadiver event graph parameters in Viskit. Event-graph parameters are initialized at setup time.

2.   State Variables

A state variable is a mathematical variable that defines an important aspect of the system. State variables change throughout the simulation and that change is called the state trajectory. The state trajectory is the graph of change in a state variable over time or "evolution of the model in time." (Buss 2000) Each state trajectory is piecewise constant and therefore only changes at events. In a typical non-moving entity simulation, most events represented on event graphs contain state changes. This is not true for tactical models where decisions and behaviors by an entity are captured. Most of the events on the Seadiver event graph do not generate state changes. In Viskit, state variables are entered and listed on the Event Graph Editor panel as depicted in Figure 22.

| State Variables |
| Double click a row to *edit*. |

name	type	description	
sensorObject	diskit.Sensor	Cookie cutter spherical sensor that detects all objects that get within sensorRange.	▲
submerged	boolean	Flag to determine if submerged or at surface.	
gpsFixNeeded	boolean	Flag to store need for GPS fix (after each mine location).	
detectionData	java.util.HashMap	Container that holds mine detection data.	
numDetectedMines	int	Number of mines detected.	
fixMoverManager	seadiver.PathDeviationMoverManager	Mover manager used during time fixes required.	
activeMoverManager	seadiver.MoverManager	Stores the active mover manager.	
wpsCreator	seadiver.LawnMowerWaypointCreator	Creates waypoints in a lawn mower pattern from a zone supplied by ZoneMap event g...	
zone	seadiver.SymetricZoneMap	Class containing zones of each mover.	
targetCollector	java.util.LinkedList	Container holding target objects detected for analysis.	
mineCollector	java.util.LinkedList	Container holder mine objects detected for analysis.	
targetDetections	int	Number of times the targets were detected.	
numberOfTargets	int	The number of surface/submerged targets in this simulation (if any).	
totalNumberOfTargetDetections	int	Stores total amount of target detections.	▼

| + | − |

Figure 22.   Seadiver event-graph state variables in Viskit. State variables can change as simulation time progresses, thus representing model state.

3.   Events

Events are one of the two fundamental components of event-graph methodology (the other being the scheduling edge). They are graphically depicted as circles on the Event Graph Editor panel. When creating an event graph, empty events are placed on the graph and then information about that event is entered into the Event Inspector that is accessed by double-clicking that event. Figure 23 depicts the event graph for the Seadiver model.
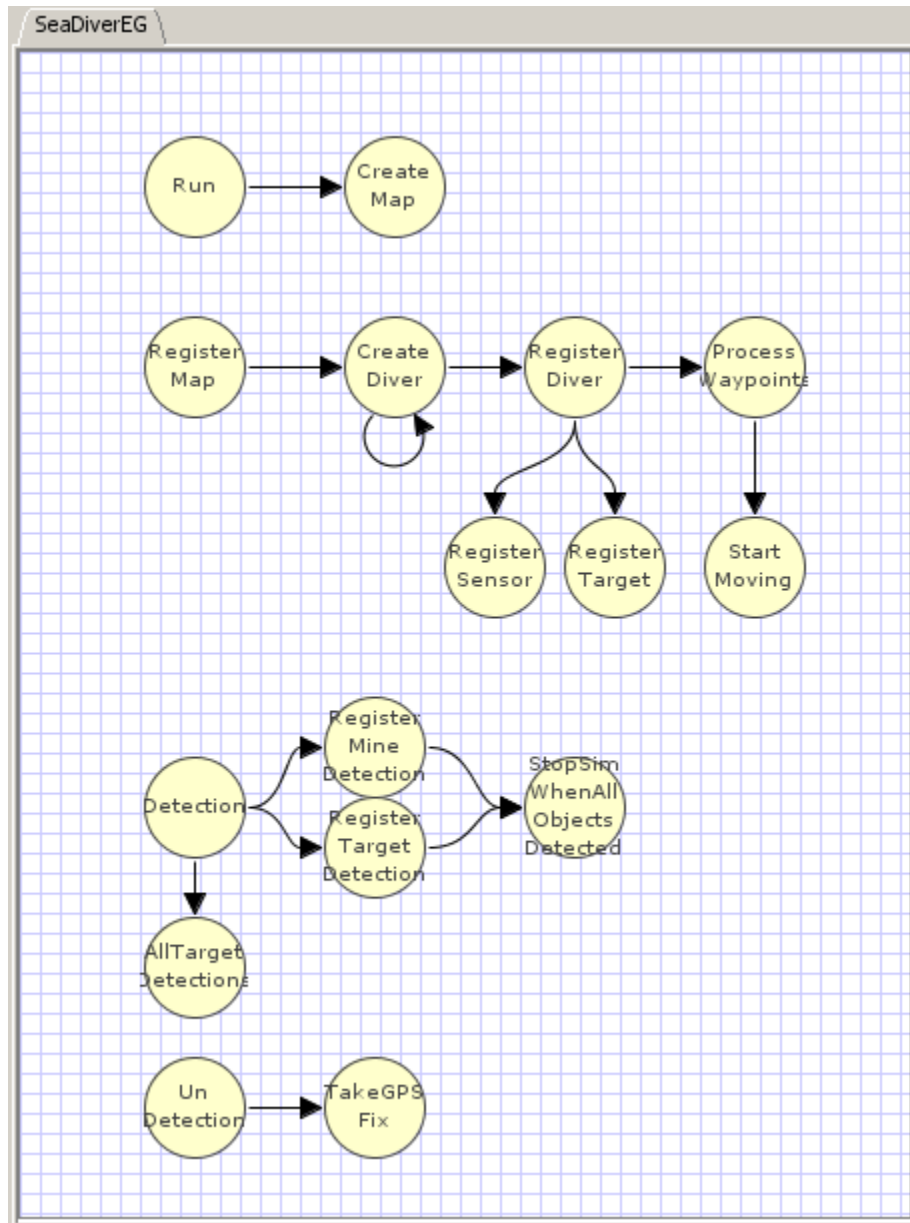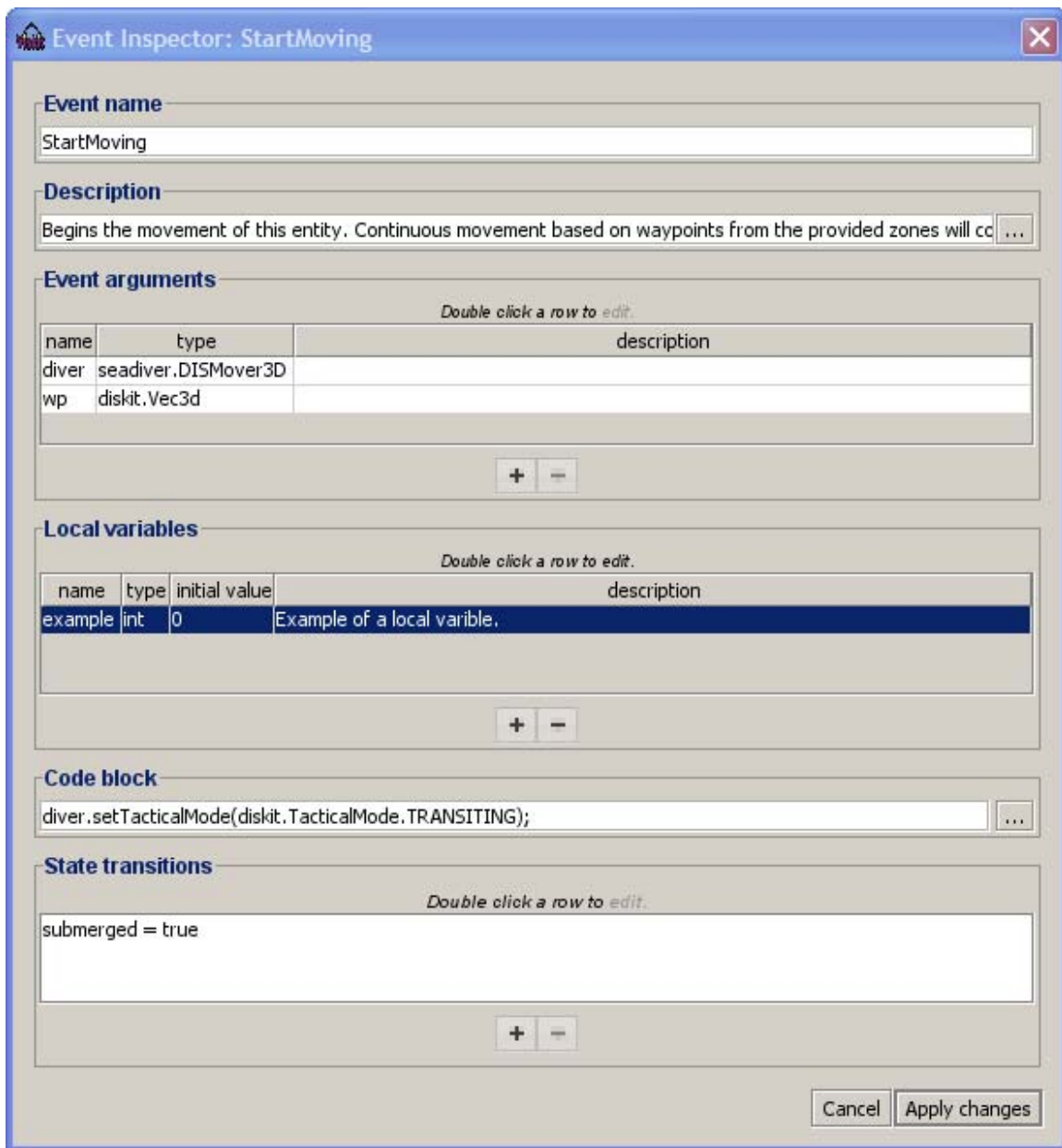
Figure 23.   Seadiver event graph shows the logical flow of information while modeling robot behaviors.

The Event Inspector is used to define events and consists of four main components: Event Arguments, Local Varibles, Code Block, and State Transitions. Figure 24 depicts the Event Inspector of the Start Moving event and its main components.

Figure 24.   Event Inspector for the StartMoving event in the Seadiver event graph.

Beginning with the Event Arguments section, the functions of the sections are explained. Arguments of events are incoming values and are directly analogous to the signature of a Java method. The composition and position of arguments determine the signature. Any call to that event from a waitDelay() method must be spelled correctly and have the exact same signature or nothing will happen. If an event has arguments then any attached upstream scheduling edges must provide the value of that argument.

All sections on the Event Inspector have plus and minus buttons used to add or remove elements. Clicking the plus button adds an empty argument and double clicking it brings up the Event Argument dialog box shown in Figure 24. The event argument dialog box is used to define the argument's name and type. Figure 25 for example depicts the event argument dialog box for ProcessWaypoints event.

Figure 25.   Event arguments dialog box for ProcessWaypoints event.

The Locals Variables section provides a location to define variables whose scope is limited to that event. They can be defined for any function, but are typically used to supply values to scheduling edges without referencing the original object. Local variables are added by clicking the appropriate plus button in the local variables section. Double click the new entry to define a new local variable in the resulting Local variables dialog box that appears. The new variable is defined by its name, type, and initial value as shown in Figure 26 which is of a Seadiver event local variable.
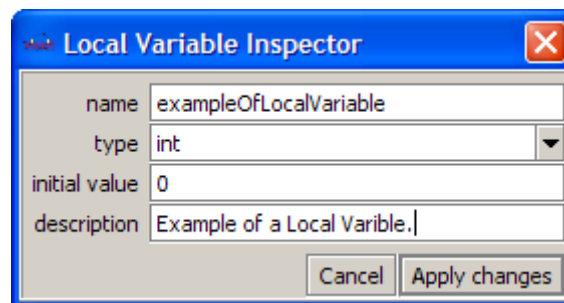
Figure 26.   Local Variables dialog box for a Seadiver event.

The Code Block section in the Event Inspector is a free form code entry area. It is used to enter any required code whose function cannot be performed by one of the other sections. The code must adhere to Java language programming syntax and rules. Unlike

36

previous sections, code is entered directly on the provided line or if more space is needed, in the box accessed thru the ellipse notation to the right demonstrated in Figure 27. The most common functions for code in the Code Block are print statements used for debug purposes and helper classes for data collection.


```
Event Code Block                                                    ✕

diver.setTacticalMode(diskit.TacticalMode.TRANSITING);

diver.setMaximumSpeed(maximumSpeed);

diver.setStartPosition(new diskit.Vec3d( zone.getZone(diver.getMoverID()).getLowX(),
zone.getZone(diver.getMoverID()).getLowY(),  zone.getZone(diver.getMoverID()).getLowZ()));

//System.out.println(diver.getStartPosition());

diver.setCruiseSpeed(maximumSpeed);

diver.setDestination(wp);

//System.out.println(diver.getDestination());

diver.start();

                                                         Cancel   Apply changes
```

Figure 27.   Code Block for a Seadiver event allows insertion of special-handling source code into the Viskit-defined event graph.

The final section is for state transitions. Similar to previous sections' add and remove state transition entries with the plus and minus buttons. Double clicking an entry brings up the State Transition dialog box. In the dialog box, select the appropriate state variable that needs modification and then give it a new value directly or through a function. Note that only state variables previously entered in the Event Graph Editor are available for change. Figure 28 depicts the State Transition dialog box using an arbitrary Seadiver event having a state transition.
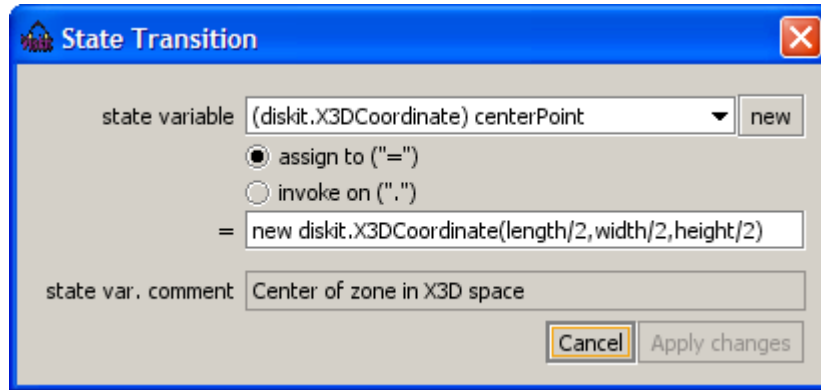
Figure 28.   State Transition dialog box for a Seadiver event.

All event graphs are basically linear programs that move sequentially from one event to the next thru scheduling edges. Most event graphs start with a Run event that initializes all state variables and resets then upon multiple simulation runs.

Following the Run event, the system moves systematically to the next event as directed by the scheduling edges. In the case of Seadiver, information is passed to the ZoneMap event graph which creates individual operating areas for each mover and passes that information back to the RegisterMap event.

From there, Seadiver diverges from typical Viskit processes when it creates the required amount of mover entities as specified by the numberDivers parameter in the Seadiver event graph. This adds complexity since each Seadiver entity is independent of the event graph. Behaviors for all Seadivers can be constructed on the Seadiver event graph but individual control is severely limited. For the purposes of this model, this was considered acceptable when many movers must be created and would have the same properties. Normally, each SimEntity would have an independent event graph.

After each mover is created, they are registered as sensors and movers with the Scenario Manager, individual waypoints are generated based on operating area, and ordered to begin movement. As the movers progress through waypoints, the Scenario Manager will manage detections and undetections. When one occurs it fires an internal 'doDetection' event that Seadiver hears (the exact mechanism is performed by

38

PropertyChangeListeners which will be discussed later) and initiates its Detection event. Based upon what type of detection is heard, follow-on events are fired down stream of the Detection event for data keeping only.

4.   Scheduling Edges

The second main component in event graph methodology is the scheduling edge. Scheduling edges connect two events together, and as their name implies, serves as a method to transition from one event to the next. In Simkit, edges are implemented by waitDelay() methods. Figure 29 depicts a Simkit waitDelay() method. The waitDelay() has four components: the scheduled event name as a String, the time delay from completion for scheduling (source) event to scheduled (target) event, the priority of events if there are two or more on the event list scheduled at the exact same time, and the target event parameters.

```
if ((contact.getMoverID() > 399) && (!targetCollector.contains(contact))) {
    waitDelay("RegisterTargetDetection",0.0,Priority.DEFAULT,(Object)(contact));
}
```

Figure 29.   Simkit waitDelay() method.

In Viskit, the waitDelay() method and therefore the scheduling edge is depicted by an arc ending in an arrow from the source event to the target event in the Event Graph Editor. The edge is defined in the Edge Inspector dialog box accessed by double clicking the graphical edge. Figure 30 depicts the Edge Inspector and illustrates that the four components of the waitDelay() method are implemented.

Figure 30.   Edge Inspector.

Figure 30 is the scheduling edge in the Seadiver event graph (Figure 23) that connects the Detection event and the RegisterTargetDetection event. As shown, the source and target events are listed, the priority is selectable from seven preset enumerations and is Default in this example, the time delay is settable to any number or function, and the passed parameter is definable and is a Mover3D called contact.

One additional property of the scheduling edge is the conditional expression. It lists the conditions that are required to be met prior to the target event being scheduled. This determines if the edge will schedule the target event. In this example, the RegisterTargetDetection event will only be scheduled if the contacts ID number is greater than 399 and has not previously been placed in the container.

40

D. ASSEMBLY EDITOR – CREATING A SIMULATION

Viskit defines a construct called the assembly that it uses to create the simulation. An assembly is constructed in the Assembly Editor panel of Viskit. An assembly is a collection of event graphs and the connections between them called SimEventListeners. The relationship between event graphs and the information passed between them via the SEL defines the foundation of the simulation. Figure 31 depicts the Assembly Editor panel with a Seadiver mission assembly open for editing.
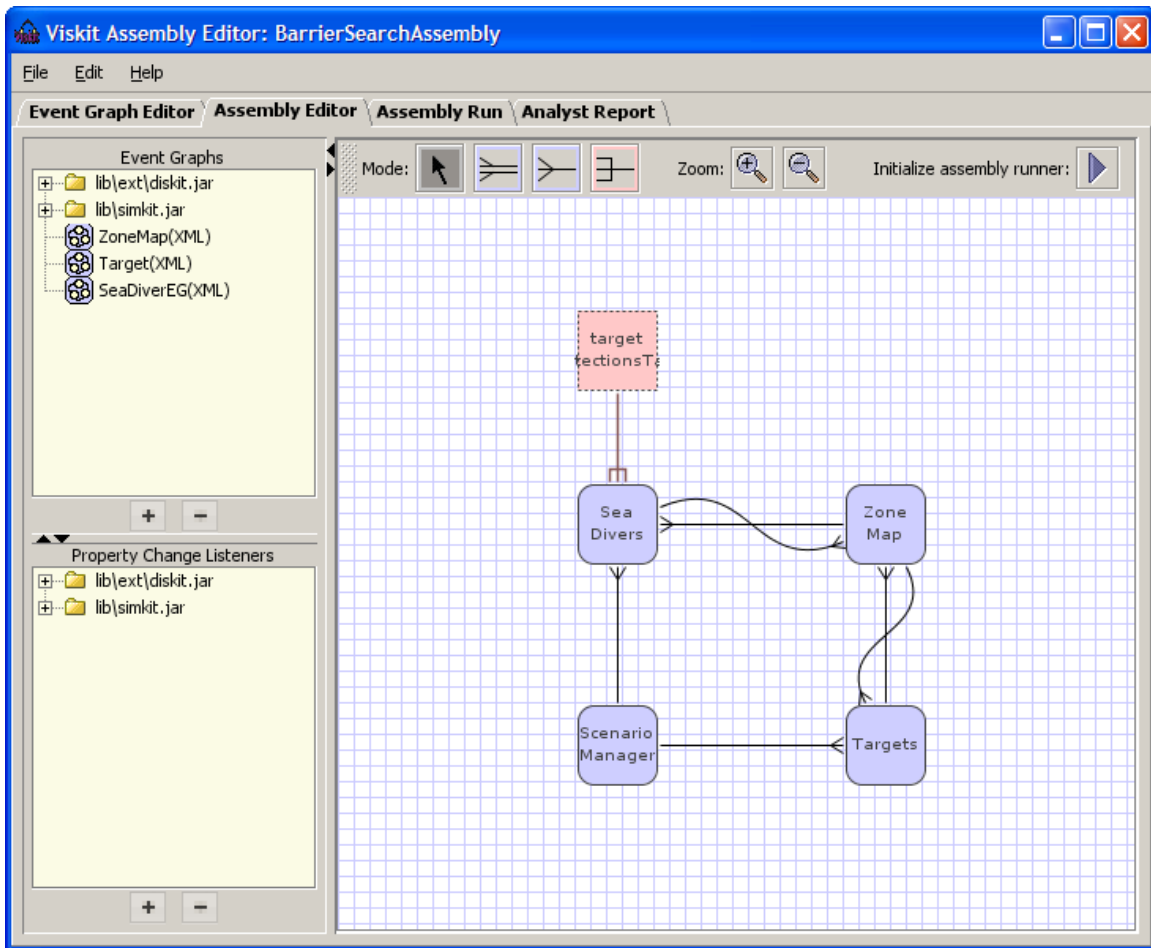
Figure 31.   Assembly Editor panel.

1. Scenario Manager

The Scenario Manager is a required element of simulations utilizing Diskit components. It provides all the functionality required to implement movement and

detection as well as the DIS protocol. This allows the modeler to create movers with sensors easily by connecting the Scenario Manager and the mover event graph with a SimEventListener (displayed in Figure 31 as the line with a small cup at the end symbolically representing an ear listening to the event graph). Additionally, implementation of the DIS protocol enables the simulation to publish DIS packets to a network enabling distributed simulation and graphics.

The parameters of Scenario Manager are listed in Table 1. Parameters are accessed through a dialog box called the Event Graph Inspector by double clicking the event graph representation in the Assembly Editor panel. The speedScale parameter is used to modify the speed of the simulation necessary when viewing the output of the simulation in a virtual environment. The clearOnReset parameter enables the statistical values to be reset for each repetition of the simulation allowing for correct reporting of confidence intervals. The last four parameters deal with details of the DIS protocol and the network interface.

Parameter	Type	Description
speedScale	double	Determines the speed of the simulation in a X3D viewer.
clearOnReset	boolean	Enables the statistical values to be reset for each repetition of the simulation
multicastIPAddress	String	The IP address of the network interface on the host computer used to transmit DIS packets.
port	integer	The port of the network interface on the host computer used to transmit DIS packets.
siteID	integer	DIS protocol setting.
appID	integer	DIS protocol setting.

Table 1. Initialization Parameters for Scenario Manager.

2. SimEntity

Once even graph models of SimEntities and objects have been created either in the Event Graph Editor or as native Simkit Java classes, they then can appear in the event graphs section of the Assembly Editor. If they appear in this list then they meet the requirements of Viskit and can be used in the assembly o create a simulation. To use the event graph, simply drag and drop it onto the assembly to create an instance of it as represented by a SimEntity Node. Figure 32 depicts an assembly of a Seadiver mission with the event graph library to the left. Note that Scenario Manager is not an event graph, but can be accessed and used in the assembly since it is a Java class that extends SimEntityBase.
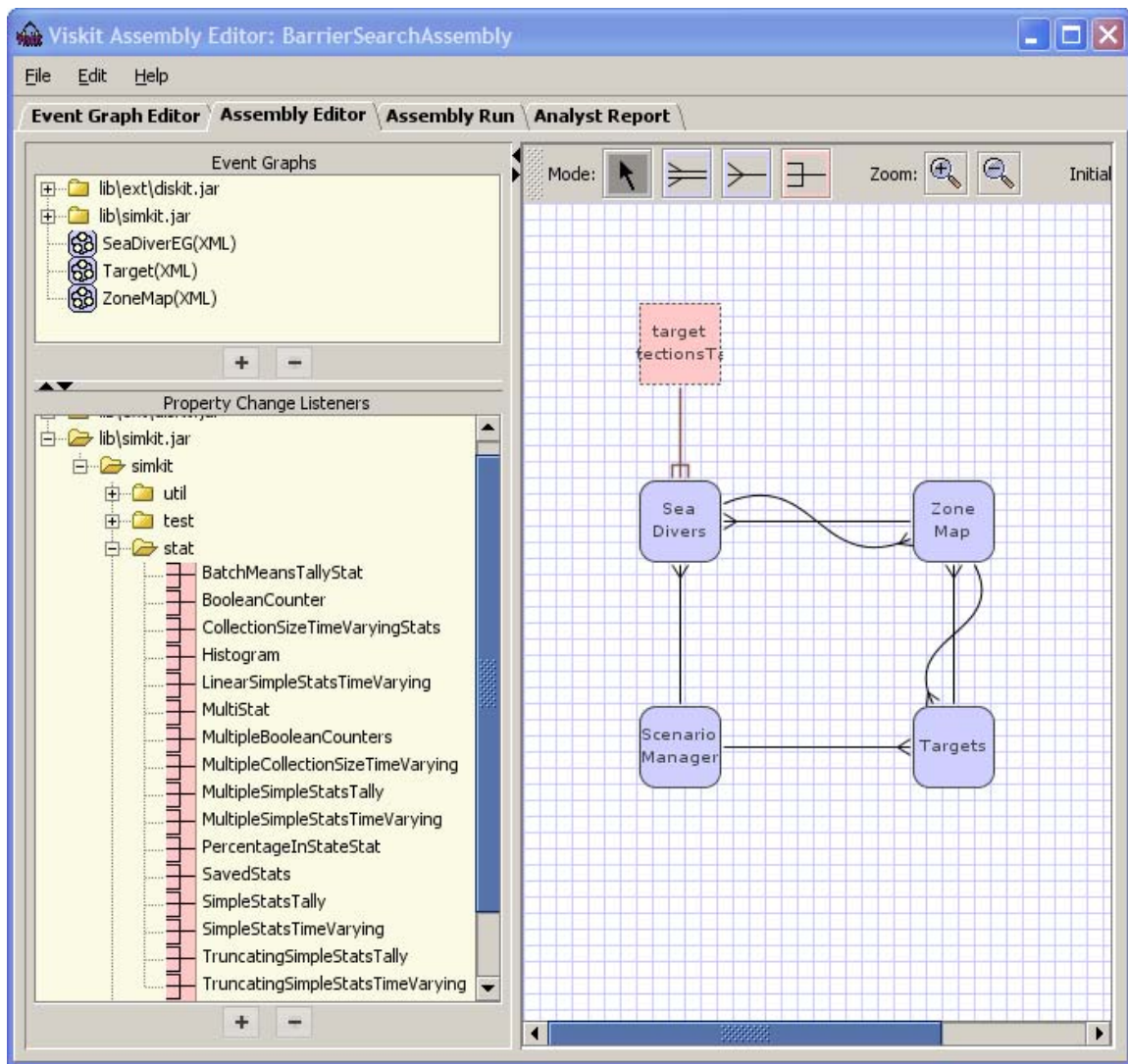
Figure 32.   Seadiver assembly depicting the model library to the left.

Several event graphs and multiple helper classes were created for the Seadiver thesis. Seadiver thesis assemblies are designed around Seadiver missions and several assemblies have been created, one for each mission. All assemblies use the Seadiver model and the ZoneMap object and either the Minefield or Target event graph. The SimEntity nodes can be named anything, but for the purpose of Seadiver thesis their names remain similar to the actual event graphs.

3. Parameter Entry

As discussed previously, event graph parameters are values that will not change throughout the simulation and are set at runtime from entries in the SimEntity nodes of an assembly. Parameters are accessed via the Event Graph Inspector dialog box by double click. Parameters can be simple numerical values like integers or doubles, or can be any other Java function that returns an object or value. The following section describes the parameters for the event graphs used in the Seadiver thesis to illustrate this functionality.

4. SimEventListener

Lines connecting SimEntity nodes in the Assembly Editor represent Simkit constructs called SimEventListeners. SimEventListeners connect two SimEntity nodes together and allow them to share information between them. To connect nodes via a SimEventListener connection, each event graph must contain an identical event (same name and signature). The source event fires then as a result the target event is fired. This has the effect of one event listening to the other event, hence the name SimEventListener.

SimEventListener connections are very beneficial to the Simkit methodology of creating simulations because they allow for passing of information between event graphs. This enables the componentization or breaking up of complex event graphs into small chunks of functionality that allow for extensive re-use and simpler debugging. The concept benefits are therefore similar to the use of inheritance.

An example of events taking advantage of SimEventListeners is the CreateMap events in both the Seadiver and ZoneMap event graphs. Figure 33 depicts the SimEventListener Connection dialog box that supplies information about a connection. In this case, the event in Seadiver passes the number of Seadiver movers that will be created and the first ID number that will be used to the ZoneMap event graph. It uses this information to create individual operating areas for each Seadiver mover based on this information and its own parameters (that determine the total size of the area). This passing of information is enabled by SimEventListener connections and would be difficult to accomplish otherwise.
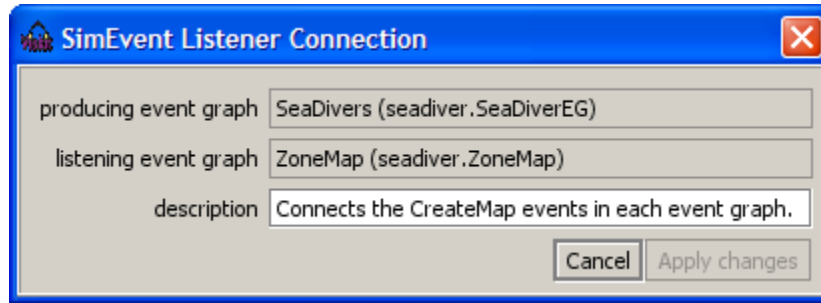
Figure 33.   SimEventListener Connection dialog box of connection from Seadiver
node to ZoneMap node in the xxxx assembly.

5.   Property Change Listener (PCL)

The Property Change Listener (PCL) is similar to the SimEventListener in that it is a construct that can be programmed to listen to a SimEntity node.   Unlike the SimEventListener connections that listen for events, PCL connections listen for changes in state variables.   By convention in Simkit, every time a state variable changes, a method called firePropertyChange() is initiated that has the effect of broadcasting that change to the simulation environment.   Figure 34 depicts the firePropertyChange() method of the Run event in the Seadiver event graph.


```
firePropertyChange("targetDetections",targetDetections);
```

Figure 34.   The firePropertyChange() method of the Run event in the Seadiver event
graph.

PCL connections are created to listen for specific state variable changes then perform preset operations with them.   Typically, these operations are for the collection, calculation and display of statistics.   Simkit has a number of built-in data collection and analysis objects that can easily be incorporated into a simulation.   Figure 31 depicts the Assembly Editor panel for a Barrier Search assembly.   In the bottom left display resides the expanded list of included PCL connections and the assembly depicts one PCL connection node (colored pink) called Target Detections Total.   New data collection objects that implement PCL connections can be created and easily incorporated into a Viskit simulation via the plus button.

46

To incorporate a PCL into a simulation, a PCL is selected from the list (or created in Java if one in the list does not fill all requirements) based on the type of property being collected such as an integer or a collection and the property's state as a function of time. Once the appropriate PCL is selected, simply drag and drop it onto the assembly and connect it to the SimEntity node with the correct state variable. Finally, select the appropriate state variable from the list accessed by double clicking the connector as shown in Figure 35.
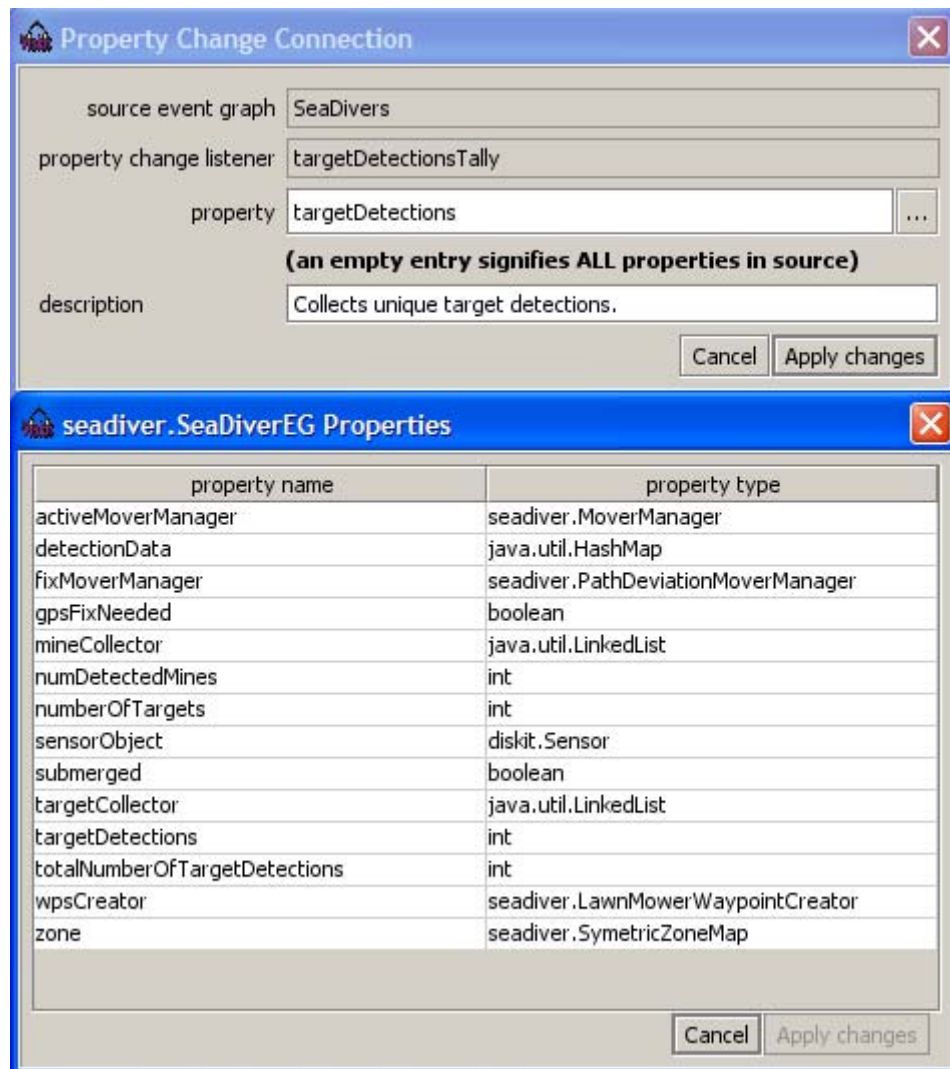
Figure 35.   Property Change Connection dialog box with state variable list expanded.

E.   MODELING FOR TACTICAL SCENARIOS

Movement is essential to the Seadiver simulation since its primary intent is to simulate the changing spatial relationship between Seadiver entities and other moveable objects such as surface/submerged targets or mines. Traditionally, DES has been used for simulating non-moving relationships as in queuing theory. This excludes simulations requiring movement and is due to the mistaken belief that the discrete event paradigm is impractical for this use. Additionally, it has been shown that in some cases using DES is desirable to other methods such as the time-step world view. (Buss and Sanchez, 2005)

1.   Movement

Movement in this thesis is accomplished by object event graphs such as Seadiver implementing the Mover3D interface (or extending a class that implements Mover3D such as DISMover3D). Implementing Mover3D ensures that the proper methods are included to enable movement functionality.

The Seadiver simulation exclusively uses DISMover3D for 3D movement capability. DISMover3D is simply an extension of 2D movement in the Simkit API. (Buss and Sanchez, 2005) provide a basic explanation of movement and detection in Simkit. In its simplest form, an object starts at position one at time one with a specific velocity. Using well known position functions, these three variables are all that is required to calculate future positions.

In Simkit, the initial position and calculated final position are modeled as events called the StartMove and EndMove events respectively. Every movement from one position to another begins with a StartMove an EndMove event as depicted in Figure 36. As shown, every StartMove event schedules a corresponding EndMove event with a time delay of that amount of time required to move from one position to the next based on the objects speed and distance between locations. If there are additional movement requirements, the EndMove event once fired will reschedule another StartMove event immediately. This process continues moving an object through locations as long as the movement requirement remains the same.
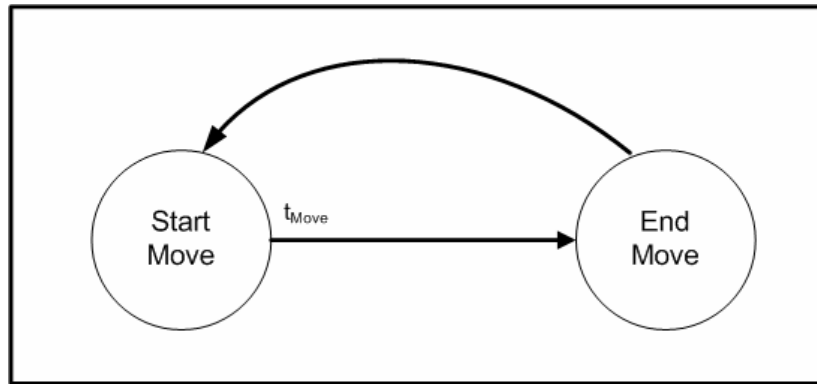
Figure 36.   Basic movement process in Simkit.

Entity movement is managed by an appropriately named Mover Manager. Mover Managers control the scheduling of StartMove and EndMove events. Mover Managers enable dynamic changing of entity behaviors such as changing from an object following a preset waypoint list to avoiding an obstacle or loitering in an area. This is accomplished by providing a mechanism to dynamically change the Mover Manager based on preset conditions.

Simkit and Diskit provide several previously created Mover Managers that can br readily incorporated into simulations. Additionally, Mover Managers can be created to meet additional movement requirements such as the Path Deviation Mover Manager in this thesis. Table x provides a summary of available Mover Managers.

Path Mover Manager	Moves an entity through waypoints.
Random Mover Manager	Moves an entity though random waypoints of which the distribution can be set.
Avoidance Mover Manager	Intermediate mover manager which moves an entity around an obstacle at a preset standoff distance between preset waypoints. The initial mover manager is restored once past the obstacle.
Intercept Mover Manager	Intermediate mover manager which moves an entity from a preset path to an intercept position. The initial mover manager is restored once intercept complete.

Zone Mover Manager	Moves an entity through preset zones of operation based on probability or A-Star search (best path) algorithms.
Path Deviation Mover Manager	Intermediate mover manager which interrupts a preset path causing the entity to dwell in a location for an amount of time. The time duration is of a preset distribution.

Table 2.   List of Mover Managers available in Simkit and Diskit.

2.   Detection

Implementation of entity behaviors that require interaction between entities cannot be achieved by movement alone. A method is required to allow entities knowledge of other entity positions, its environmental limitations, and to react to them. This is accomplished in Simkit and Diskit by the sensor implementation.

The simplest and most practical sensor is the cookie cutter sensor implemented in Simkit. Figure 37 depicts the notional concept of the sensor. In this example, there is one 2D sensor attached and coincident to entity A. The sensor moves with entity A and has a radius R in which other entities will be detected if the circle prescribed by R around entity A is entered. Similarly, when the entities achieve a distance greater than R apart, the sensor does not detect the other entity.
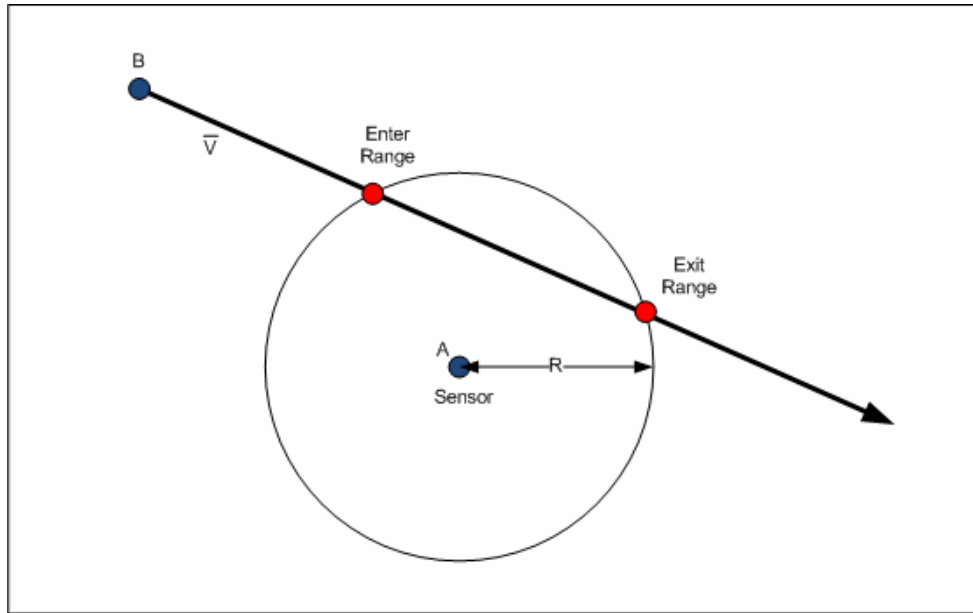
Figure 37.   Cookie-Cutter sensor. The basic scenario.

In Simkit and Diskit, this process is modeled by the Detection and UnDetection events.   Figure 38 depicts the DES methodology for the sensor Detection and UnDetection events. These events correspond to the instance one entity enters within the radius of a sensor and is projected to leave that radius respectively.   Specifically, Detection events are placed on the event list when it is calculated that entity B will enter range of the sensor. Similarly, an UnDetection event is scheduled at the same time when entities change velocity vectors, recalculation of enter and exit range values and rescheduling of Detection and UnDetection events is performed.
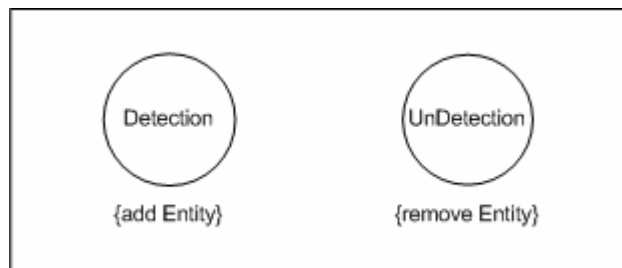
Figure 38.   Sensor DES methodology.

The Seadiver model implements the spherical cookie cutter sensor provided by Diskit.   It expands the 2D cookie cutter sensor in Simkit to three dimensions. Functionally it operates the same as the sensor described above. These sensors can be

extended to provide advanced functionality such as implementation of non-circular range or detection ranges based on probabilities, but the spherical cookie cutter sensor is adequate for this proof of concept thesis.

F.   STATISTICAL RESULTS

The output of a stochastic simulation is typically statistics of static or time-varying nature. Each run of a simulation produces another set of statistics called a repetition. A number of repetitions are produced which are used to generate confidence intervals. Confidence intervals are then used to analyze the model for expected behavior and to generate logical inferences from unexpected behavior.

The Assembly Run panel is the location where each repetition and calculated confidence intervals are viewed. Section C of Chapter III discussed the basic functionality of the Assembly Run panel. Additionally, it is the location where the simulation is initiated and its control settings adjusted. Table 3 lists the simulation settings and their descriptions.

Setting	Description
Sim Stop Time	Determines when the simulation will stop. Simulation can also be stopped programmatically based on preset events.
# Replications	Determines the amount of replications the simulation will complete. This automatically allows for computation of confidence intervals for stochastic simulations.
Verbose Output	Determines how much information is written to the output panel on the upper right. Primarily used for simulation debug purposes. If verbose is selected, every entry in the event list is written to the output panel.
Save Replication Data	Determines if the statistical data output of the simulation is written to file. This file is in XML format and is used in generation of the Analyst Report.
Print Replication Reports	Determines if the statistical replication data output of the simulation is written to the output panel.
Print Summary Reports	Determines if the summary report is written to the output panel.

Table 3.   Assembly Run panel settings and descriptions.

G. SUMMARY

This chapter described in detail how to create a DES simulation in Viskit. Using this chapter as a guide, the reader has the basic knowledge required to construct a basic event graph in Viskit's Event Graph Editor and then integrate that event graph into a simulation in the Assembly Editor. Additionally, this chapter provided the conceptual framework for implementing moving entities with detection capability and why DES is a suitable framework for such.

THIS PAGE INTENTIONALLY LEFT BLANK

V. TACTICAL CONSIDERATIONS FOR SIMULATION GOALS AND REQUIREMENTS

A. INTRODUCTION

This chapter discusses the logic and methodology behind why modeling and simulation was used to analyze the Seadiver UUV. First, the benefits and advantages of modeling and simulation are previewed for the design and construction of complex robots. The process of creating the two simulated missions of this thesis is then explained in detail. Finally, the communication requirements for UUVs performing these types of missions are described.

B. NEEDS AND REQUIREMENTS FOR ROBOT MODELING

Modeling and Simulation (M&S) is beneficial and probably essential for the creation of complex moving entities such as autonomous UUVs. It enables realistic evaluation of scenarios of interest while identifying discrepancies and deficiencies prior to their becoming problematic in the construction or testing phases. The following sections describe how creating a model optimizes production while minimizing risk, cost, and time to deployment.

1. Robot Design and Construction

Creation of a 3D model during design and construction allows for accurate identification of physical characteristics of the UUV. Characteristics such as dimensions, density, weight, and center of gravity and buoyancy are now exposed for modification. Designers can experiment with alternate products and materials to determine how they affect the system as a whole. An example would be changing the structural material from steel to a composite and its affects on properties such as the center of gravity or buoyancy. In effect, modeling and simulation enables shifting a typically iterative process from the construction phase (when errors are costly or fatal) to the design phase (when they are more easily addressed).

55

2. Predicted Dynamics Response

Modeling enables determination of a vehicle's predicted dynamic response. This allows for accurate estimation of operational characteristics such as speed and turning radius which in turn enable insight regarding how a particular UUV might be employed.

3. Robot Control and Mission Planning

One complex issue inherent in all UUVs is the method of operator planning and then transferring a mission to a robot for execution. The mission itself must be capable of achieving the desired objectives and be in a format the robot can understand. Modeling and simulation enables mission construction in a 3D virtual environment such as AUVW. This allows for visual and interactive mission planning producing the best case scenario for completion of mission objectives.

A mission is constructed and saved in AUVW as an AVCL file. Using an open-standards based XML file language such as AVCL allows for simple conversion and mapping to a proprietary robot command language through use of an XSL transformation. A single virtual simulation program such as AUVW can now comparably plan and control many diverse robots, even when each utilizes a different proprietary command language.

4. Sensor Characteristics

There are a multitude of sensors available for robot use. Determining which sensor (or combination of sensors) is acceptable for a certain mission can be difficult. Modeling and simulation allows for evaluation of various sensor characteristics and packages prior to deployment which increase the probability of a successful mission.

5. Power Budget

A major driving factor behind UUV design is endurance. Endurance is controlled by the power budget which is affected by the power requirements for propulsion (buoyancy control for the Seadiver), communications, and sensors. Modeling and simulation can optimize the endurance robots by enabling the selection of the best combination of components.

C. CHALLENGING TACTICAL SCENARIOS

This thesis researches the ability of robots with long endurance to perform two historically difficult naval missions: Minefield Search and Barrier Search. According to the U.S. Department of the Navy's Mine Warfare Plan,

> The sea mine remains today- as it has throughout history- an exceptionally powerful and cost-effective tactical weapon that deserves a prominent position within any naval arsenal. (Johnson and Jones, 2000)

Indeed, many navies around the world maintain the capability to utilize sea mines for littoral protection/denial or sea lane denial. Mine warfare is a serious threat to U.S. power projection. Mines have been responsible for seriously damaging 14 U.S. Navy ships since the Korean War. In a single 3 year stretch from 1988 to 1991, 3 U.S. Navy ships hit mines resulting in over $121 million in damages while the total cost of the mines were $13,000. (Goure 2002)

Mines are problematic for the U.S. Navy not only due to their low cost and widespread proliferation, but also because of their difficulty in being located and removed. There is a need to perform mine countermeasures operations quickly and covertly without exposing assets to potentially hostile action. Removal of mines remains time intensive, highly observable, and requires the use of many assets.

Barrier search is another particularly difficult mission. Barrier search is defined in this thesis as the covert detection and location of enemy vessels, either surfaced or submerged, over a large area. Early detection and warning of the presence of hostile forces such as submarines is an essential prerequisite for many of the advanced operational concepts of operations. Additionally, a barrier search mission can be employed to locate enemy combatants exiting port or converging on an attack location. Thus mine warfare and barrier search are significant challenges which might reveal special value in a SeaDiver glider UUV.

1. Minefield Search

The first mission simulated by the Seadiver model is minefield search. It endeavors to determine if many Seadiver robots can operate collectively to effectively search a notional minefield. The results of the simulation are then intended to be

processed to determine metrics for use in analysis either against existing mine-countermeasure solutions or also to re-initialize the simulation with modified parameters to produce an optimized solution.

The notional minefield used for this research is a shallow area of length, width, and depth dimensions. A select number of mines are dispersed over the area in configurable 3D distributions along the length, width, and depth. In effect, the product is a 3D rectangular volume with mines dispersed randomly inside.

The minefield is then divided into separate Seadiver operation zones, one per UUV. When the simulation begins, all robots start searching their respective zones in a lawn-mower search pattern depicted in Figure 39. Each parallel leg is at a distance apart equal to the sensor range. Such an exhaustive cooperative-search pattern ensures 100 percent coverage of the area in the length and width dimensions.
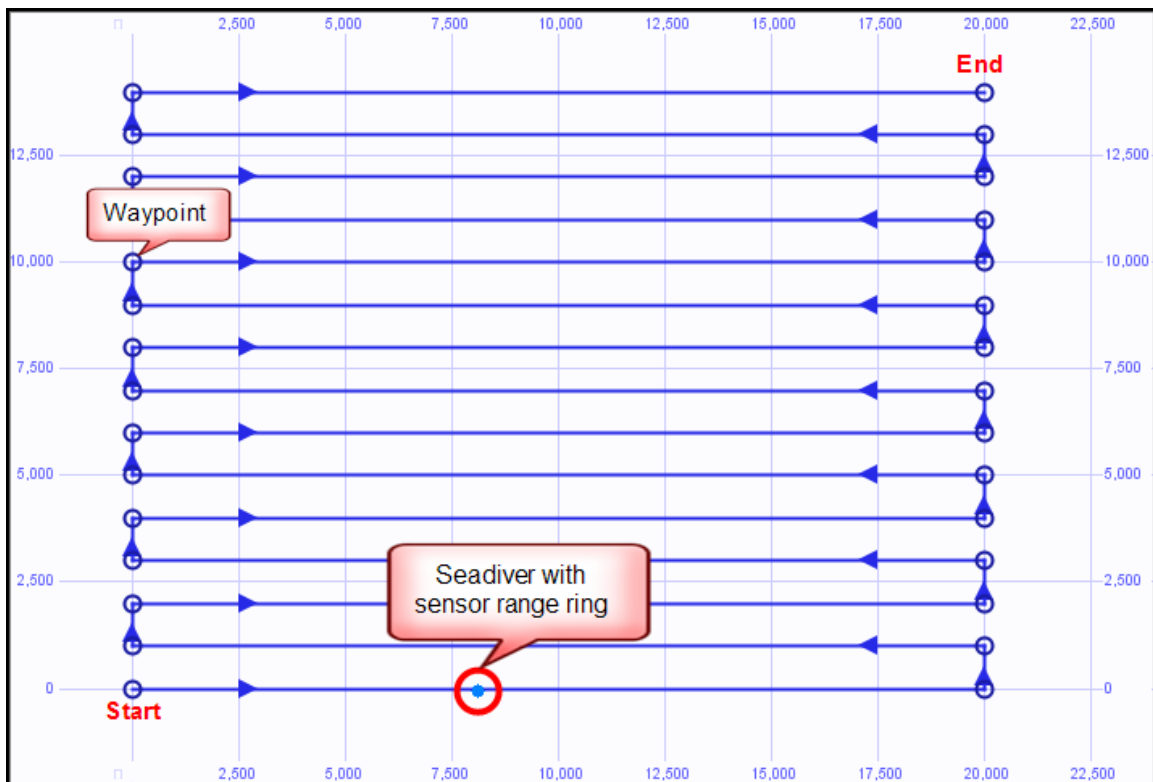
Figure 39.   Seadiver search pattern over a 20 km by 14 km zone. Sensor range is 500 meters.

Each mine is detected as a Seadiver moves within a specific sensor range. Upon detection, the UUV then simulates taking a GPS fix and records mine location in an internal database. Each GPS fix observation causes the Seadiver to delay at the mine location for an amount of time determined by a configurable distribution. Upon completion of the GPS fix, the Seadiver continues the search indefinitely. The simulation is configured to stop when all mines are located.

2. Barrier Search

The second mission simulated by the Seadiver model is the barrier search. It simulates an array of Seadivers spread across a notional area. It attempts to analyze the probability of this array of UUVs detecting surfaced or submerged targets traversing the area. The results of this simulation are to be used to determine if an array of Seadiver UUVs can be deployed in such a way as to effectively detect and localize an approaching hostile vessel.

The notional area is constructed and Seadivers are dispersed in the same manner as in the minefield search mission described above. Each target traverses the area from East to West bisecting randomized waypoints. The number of targets, number of waypoints, waypoint distribution, and then Seadiver and target speeds are specified as configurable parameters. Figure 40 depicts the barrier search mission setup.
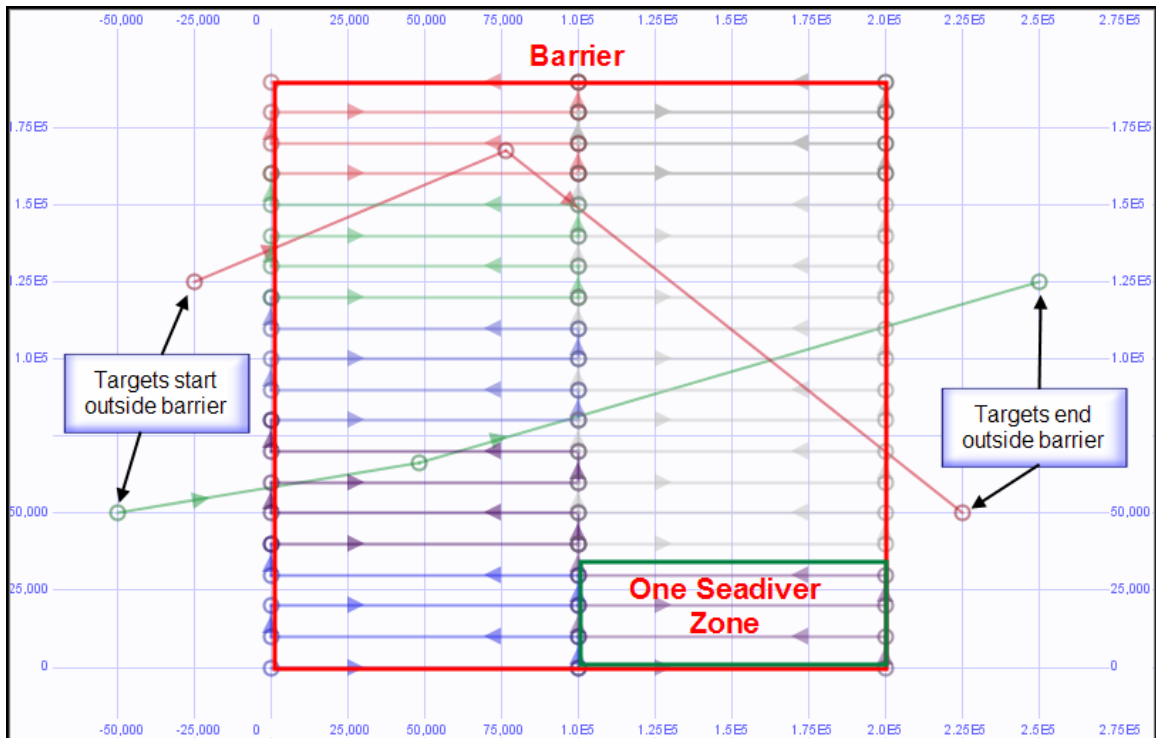
Figure 40.   Barrier search mission setup. Mission includes 10 Seadivers and 2 Targets.

When the simulation begins, each Seadiver commences the search of its zone in the same lawn-mower search pattern depicted in Figure 40 and the targets commence their traversal of the area. As targets approach within Seadiver sensor range they are detected. All detections are logged for later analysis. The simulation ends when all targets have traversed the area. Data is collected on time to detect all targets and total number of target detections.

D.   COMMUNICATIONS   PERIODICITY   CONSIDERATIONS   AND REQUIREMENTS

The ability to communicate and obtain accurate fix information is essential to these two missions. All the raw sensor data in the world is useless if it can't be placed in the hands of those who need and can act on it. Similarly, it is good to know the number of mines in a minefield, but extremely more useful to know *where* those mines are located.   Therefore it is a requirement for Seadiver UUVs to have the capability to communicate with satellites for contact reporting and GPS data, and this simulation assumes they have this capability. Table 4 outlines these requirements for each mission.

	Minefield Search	**Barrier Search**
GPS fix	1) Mine located 2) Start of each leg	1) Contact located 2) Start of each leg
Contact Report	1) Weekly 2) When commanded 3) Upon first contact	1) Contact located 2) When commanded
Status Report	1) Weekly	1) Weekly

Table 4.　　Communication requirements for each mission.

E.　SUMMARY

This chapter discussed the logic and methodology behind why modeling and simulation was used to analyze the Seadiver UUV. The benefits and advantages of modeling and simulation are presented for design and construction of complex robots. The process of creating two simulated missions for this thesis is described in detail. Finally, the communication requirements of UUVs performing these types of missions are assessed.

THIS PAGE INTENTIONALLY LEFT BLANK

VI.   SIMULATION SCENARIO DESIGN AND DESCRIPTION

A.   INTRODUCTION

Every model strikes a balance between generality and fidelity. This research is no exception. The prevailing conditions such as current design progress and required level of reuse necessitate this model to be highly general. Section B details the assumptions this model requires for this condition.

Section C details how entity behaviors are captured in this research during the creation of the four event graphs that compose this thesis. The event graphs represent three moving entities (Seadiver, Target, and Minefield) and one area knowledge object (ZoneMap).

Simulations are created in a Viskit construct called an assembly using event graphs and SimEventListener connections. Section D details how the aforementioned event graphs along with Scenario Manager were integrated into the functional simulations.   This thesis constructs two functional assemblies and therefore two simulations which mimic the ability of the Seadiver to conduct relevant military missions.

Simulations constructed in Viskit are effectively programming constructs. The output of this environment is statistical data and therefore makes it difficult to determine the validity of such data. Disparity in statistical data could be in the best case typical stochastic variation or in the worst case model inaccuracy or fault. Section E describes how AUVW is employed to validate the simulation results.

B.   ASSUMPTIONS

Every simulation must strike a balance between fidelity and abstraction (generalization) to enable observation of results of interest while at the same time ensuring those results are adequately representative of real-world conditions to promote logical conclusions. For the purpose of this thesis, this simulation is highly abstract and very general. This was necessary for two reasons. First, the vehicle has not been completely constructed yet, and therefore most of its operating characteristics can only be

surmised at this point. Second, this thesis focuses strictly on the vehicles ability to perform certain missions such as minefield or barrier search, and therefore a higher fidelity for other characteristics is not required. In that context, many assumptions were made constructing the simulation and Table 5 below summarizes them.

Assumption	Rationalization
Mover speeds	All mover speeds are constant from start to finish. Seadiver speed is set to 3 knots. Target speed is set to 12 knots.
Mover turning characteristics	Seadivers and Targets turn instantaneously.
Seadiver endurance	Seadiver is speculated to have 30 day endurance. Endurance is limited based on vehicle power requirements which include sensor suit power requirements. Longer duration may be possible.
Sensor characteristics	Cookie cutter spherical sensor used. All movers are detected when entering range and undetected when exiting range. A nominal sensor range is set as a parameter which can be changed to evaluate tradeoffs between endurance, range, and number of movers to adequately accomplish a mission.
Navigation capabilities	There is no navigational error correction required for Seadivers. Seadivers pass through all waypoints.
Neglect set/drift	Environmental factors that affect Seadiver motion are neglected for simplification in the DES. Set and drift analysis on missions can be conducted in AUVW.
Non-random starting positions.	All Seadivers start at the origin of their respective zones at position (000) relative.

Table 5. Major assumptions of the Seadiver simulation.

C.   TACTICAL DEFINITION: EVENT GRAPHS

In the context of creating a simulation based on a UUV of assumed realistic capabilities conducting undersea autonomous missions, a list of required model functionality was generated. The identified minimum requirements are:

- Entities must be capable of 3D movement along a path.

- Entities must be capable of sensing and reacting to other entities.

- Entity paths must be automatically generated based on supplied operating area.

- Key parameters of entities must be adjustable such as speed, operating depth, and time delays.

- The operating area must be variable with individual entity operating zones automatically generated.

- There must be a method to collect relevant statistics of operational characteristics.

- The simulation must be capable of creating and managing many moving entities (design goal: 100) simultaneously.

- The simulation must produce a mission specifying entity movement in AVCL format for validation in AUVW.

The following sections describe the methodology used to enable this functionality through event graph (model) and assembly (simulation) design.

1. Seadiver Event Graph
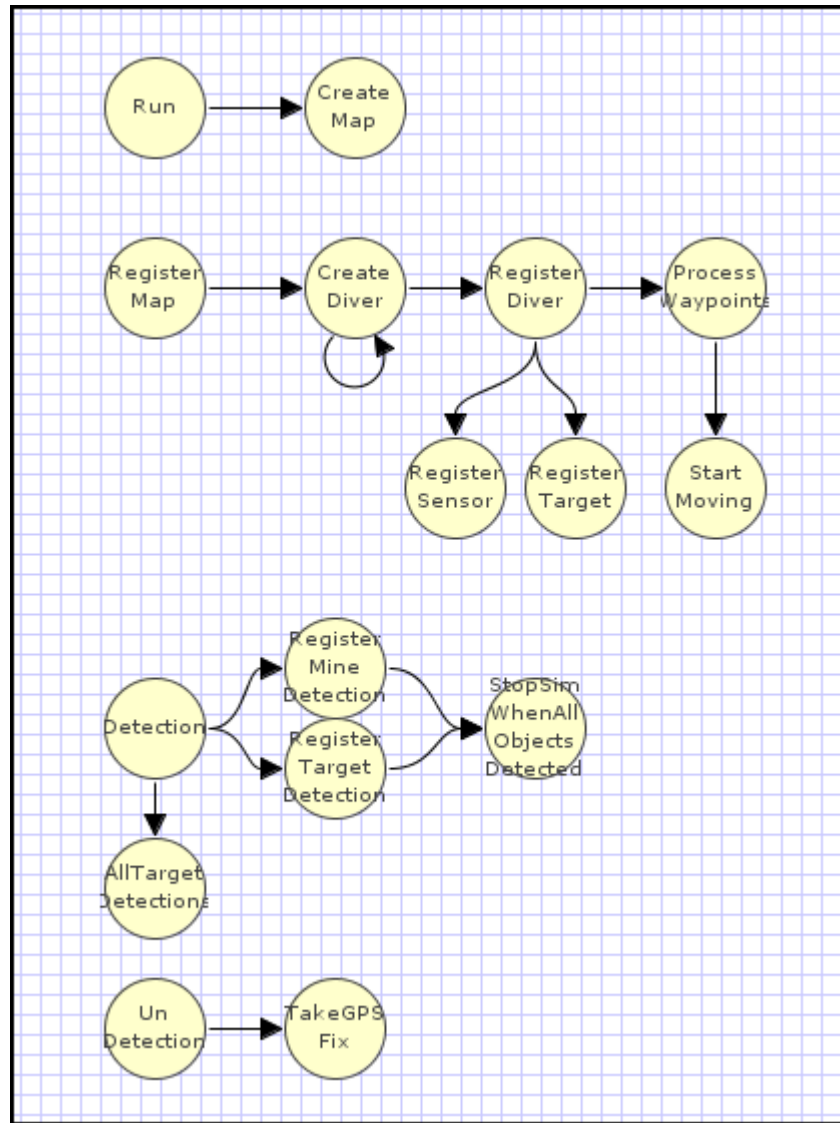
Figure 41.   Seadiver event graph.

The Seadiver model depicted in Figure 41 above is an event graph that extends SimEntityBase and hence can be used in Viskit. It is designed to model an arbitrary group of moving Seadiver UUV entities. This event graph can be divided into three separate functional sections that control the behavior of the model.

The first region consists of the Run and Create Map events. Every event graph requires a Run event by convention in Simkit. The Run event allows for initialization of each variable upon commencement of multiple repetitions. Once the event graph and its variables have been initialized, the Create Map event is fired which passes two parameters, numberDivers and firstMoverID, to the ZoneMap event graph for processing.

The second section, consisting of the events between the Register Map and Start Moving events, is responsible for proper creation and initialization of the amount of Seadiver moving entities specified by the numberDivers parameter. This process begins with the Register Map event which is fired by the ZoneMap event graph as it passes an array of zones to the Seadiver event graph. Register Map schedules Create Diver event with firstMoverID as a passed parameter which, along with the self-scheduling edge, is analogous to a programmatic FOR loop. The result of this construct is that all events downstream until the Start Moving event are conducted an amount equal to the number of Seadivers to be created.

The Register Diver event handles the actual creation of the DISMover3D entities. A unique entity is created each time Register Diver is fired. The Register Sensor and Register Target events fire, once for each moving entity, and are required to register each entity as a mover and sensor with the Scenario Manager in each assembly.

Next, the Process Waypoints event creates unique waypoints for each mover in a lawn-mower search pattern via the LawnMowerWaypointCreator class. These waypoints are then used to create an instance of pathMoverManager which controls Seadiver progression of movement through the waypoints. Finally, the process fires the Start Moving event which initiates movement for each Seadiver DISMover3D entity.

The third functional section senses object detection and un-detection, defines the behavior of the mover after those events, and provides the logic for simulating GPS fix collection. The Detection event fires every time a Seadiver entity detects another entity, including another Seadiver. The following events such as Register Mine Detection, Register Target Detection, and All Target Detection control filtering and registering individual entity detections for use in statistic gathering by iterating appropriate state variables. Detected objects that move beyond sensor range are Undetected. Upon

67

Undetection, Seadivers simulate taking a GPS fix by loitering in the area of undetection of an amount of time determined by the fixDelayTime parameter. This is accomplished by switching mover managers from pathMoverManager to fixMoverManager. Once this delay is complete, fixMoverManager is replaced by pathMoverManager and movement commences where it was interrupted.

The following sections detail the parameters and state variables for the Seadiver event graph in detail. Note that all units are in meters and seconds.

a.   *Seadiver Parameters*

Figure 42.   Seadiver initialization parameters as shown in the assembly Event Graph Inspector.

Figure 42 depicts the Event Graph Inspector of the Seadiver model. Table 6 lists and defines each parameter.

Initialization Parameter	Type, Units	Description
maximumSpeed	double, m/s	Determines the maximum speed of the mover.
firstMoverID	int	Determines the starting number for mover ID numbers. Each Seadiver will be issued an ID starting at this number as required by the DIS protocol that each entity have a unique number.
numberDivers	int	Determines how many Seadiver movers will be created and used in the simulation.
sensorRange	double, m	Determines the range of the sensor for the Seadiver. All Seadiver sensor ranges will be equal.
fixDelayTime	Random Variate, s	Determines the amount of time required to take a GPS fix. It is based on the random distribution selected in the constructor.
inputFileTemplate	String	File name and path to the template used to create AVCL mission files. Note that path is relative to the Viskit Behavior Libraries directory and forward slashes (/) are required.
outputFileName	String	File name and path to write created AVCL mission files. Note that path is relative to the Viskit Behavior Libraries directory and forward slashes (/) are required.
operatingDepth	double, m	Determines the nominal operating depth of the Seadiver.

Table 6.   Seadiver initialization parameters defined.

b.   *Seadiver State Variables*

State Variable	Type	Description
sensorObject	diskit.Sensor	Spherical cookie cutter sensor attached to each Seadiver. Any object coming within the sensorRange parameter will be detected.
submerged	boolean	Indicates if the mover is submerged or not. The only time a move is not submerged is during a fix.

State Variable	Type	Description
gpsFixNeeded	boolean	Indicates when a GPS fix is needed which is after a new detection.
detectionData	HashMap	Container used to collect detection data.
mineDetections	int	Counter to collect statistics on the number of mines detected.
fixMoverManager	seadiver. PathDeviation MoverManager	Mover manager that controls the actions of the mover when taking a fix. Specifically, the mover loiters in the same area for an amount of time equal to fixDelayTime parameter.
activeMoverManager	seadiver. MoverManager	Mover manager helper that simply holds the current mover manager.
wpsCreator	seadiver. LawnMower WaypointCreator	Generates waypoints in a lawn-mower search pattern in a specific zone and creates AVCL mission files from created waypoints.
zone	seadiver. SymmetricZoneMap	Creates equal sized operating areas for Seadiver movers, one for each Seadiver in a pattern that uniformly distributes movers over an entire area.
targetCollector	LinkedList	Collection that holds unique target detections. Used to determine if a target has been previously detected.
mineCollector	LinkedList	Collection that holds unique mine object detections. Used to determine if a mine has been previously detected.
targetDetections	int	Counter to collect statistics on the number of mines detected.
numberOfTargets	int	The number of targets created by the simulation. Passed parameter from the target event graph. Used by targetCollector to determine if all movers were detected.
totalNumberOfTarget Detections	int	Counter to collect statistics on the amount targets were detected. These are not unique. Used to determine how many times an individual target was detected.

Table 7.   Seadiver State Variables.

2.   ZoneMap Event Graph

Figure 43 depicts the ZoneMap event graph. ZoneMap is a helper object that collects information on the size of the operations area and uses it to provide info to the other SimEntities such as Seadiver and Target. This info is required for the simulation and used to perform tasks such as providing individual operating areas for each Seadiver or waypoint generation for Targets. It also is broken into three functional areas.
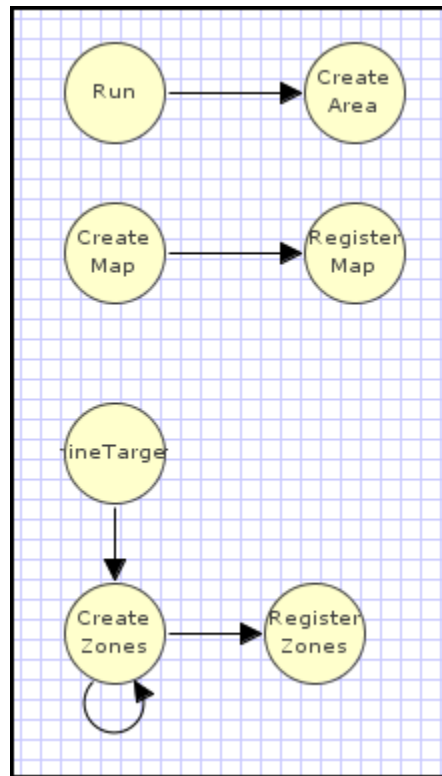
Figure 43.   ZoneMap event graph.

The first section controls initialization of the ZoneMap SimEntity. It consists of the Run and Create Area events. The Run event is responsible for initialization and reset of event graph state variables each repetition. The Create Area event creates the total operating area for the simulation based on the parameters length, width, and height. It stores this area as a diskit.ZoneGeometry object that exposes methods to work directly with the area.

71

The second section consists of the Create Map and Register Map events. It contains the functionality required by the Seadiver event graph. The Create Map event receives the number of Seadiver entities as an initialization parameter from the Seadiver event graph and creates individual zones for each Seadiver entity using the seadiver.SymmetricZoneMap class. This class divides the total zone into a number of zones equal to the number of Seadiver entities. It uses 100 percent of the area with no overlap and keeps the number of movers along the length and width equal if possible. Its algorithm works best with a large number of Seadivers. Figure 44 below is a representation of an area 100 by 60 kilometers long, sectioned into 24 individual zones corresponding to 24 individual Seadivers.
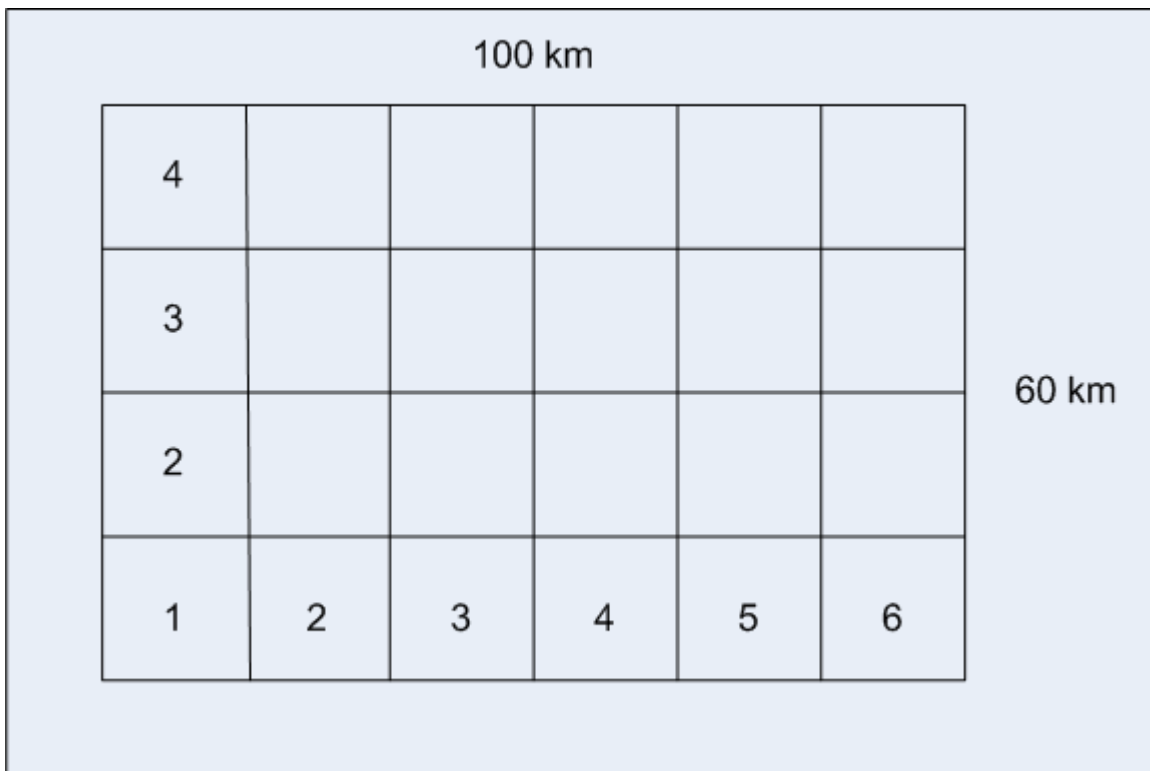
Figure 44.   Notional operating area sectioned into individual operating zones, one for each Seadiver.

The third section consists of the events Determine Targets, Create Zones, and Register Zones. This section creates zones for the Target event graph use in creating waypoints. The Determine Targets event receives the number of target entities that are created and the number of waypoints that are created for each target from the Target

72

event graph. Using this information, it sections the total area into equally sized zones. In the Create Zone event, the zones are created for each target in an amount equal to the number of waypoints each target can navigate through. The zones are individually added to a collection (zoneList) that is passed to the Target event graph in the Register Zones event. Figure 45 represents an arbitrary area sectioned into individual waypoint zones.
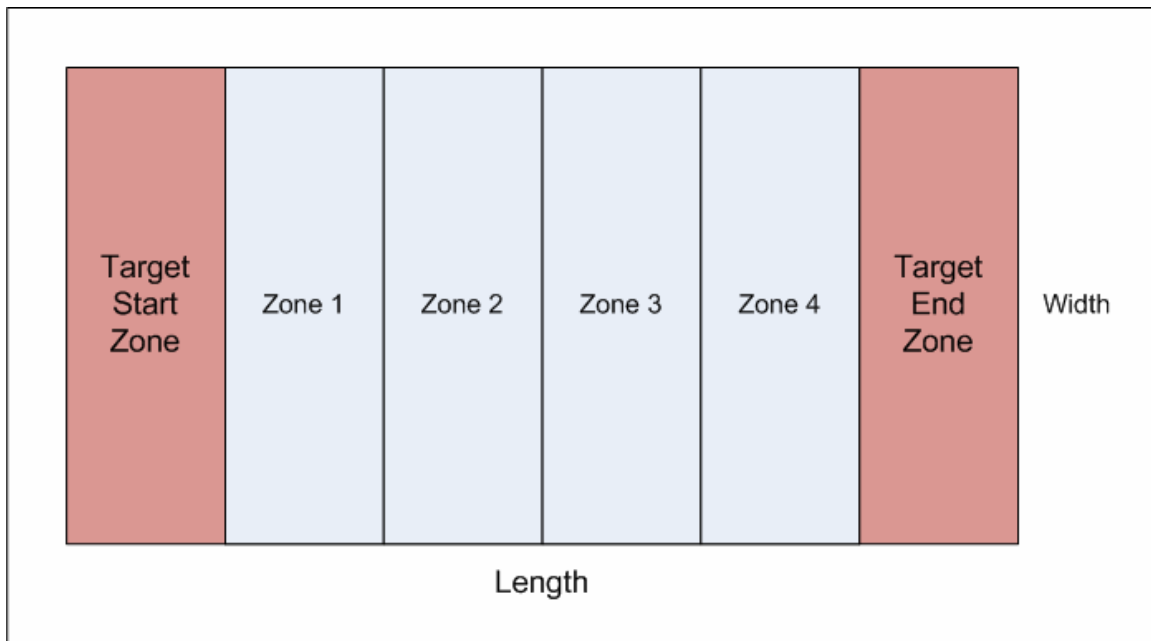
Figure 45.   Operating area sectioning performed ZoneMap for the creation of waypoints in Target event graph. The blue area is the total operating area. The red area indicates where targets start and end traversal of area (outside area).

Figure 46.　　ZoneMap parameters as shown in the assembly Event Graph Inspector.

Figure 46 depicts the Event Graph Inspector of the ZoneMap object. These are the parameters defined to enable specific functionality of the model while maintaining an adequate level of generality.　Table 8 lists and defines each ZoneMap parameter.

Initialization Parameter	Type, Units	Description
origin	X3Dcoordinate	Determines the origin or reference point for the coordinate system used by the simulation.
length	double, m	Determines the length (x value) of the area.
width	double, m	Determines the width (y value) of the area.
height	double, m	Determines the height (z value) of the area.

Table 8.　　ZoneMap initialization parameters defined.

ZoneMap State Variables

State Variable	Type	Description
moverID	int	Passed parameter from Seadiver event graph. Used to match created zones to movers.
numberOfTargets	int	Passed parameter from Target event graph. Passed to Seadiver event graph for use is statistics collection.
centerPoint	diskit. X3DCoordinate	A 3D location generated from individual zone dimensions. Used in the construction of zone object.
zones	seadiver. SymmetricZoneMap	Creates an equal and symmetric amount of zones from the total area bases on the amount of Seadiver movers.
axisAngle	diskit.Vec4D	Variable used to construct the area state variable. Used to determine rotation about the vertical axis of the zone.
area	diskit.ZoneGeometry	A 3D volume object which determines the individual operating areas for each Seadiver.
numberOfWaypoints	int	Determines the amount of waypoints created for a target as it navigates through the operating area. Passed parameter from Target event graph.
zoneList	LinkedList	Collection object used to hold all created ZoneGeometry areas to be passed to Seadiver.

Table 9.   ZoneMap State Variables.

3. Minefield Event Graph
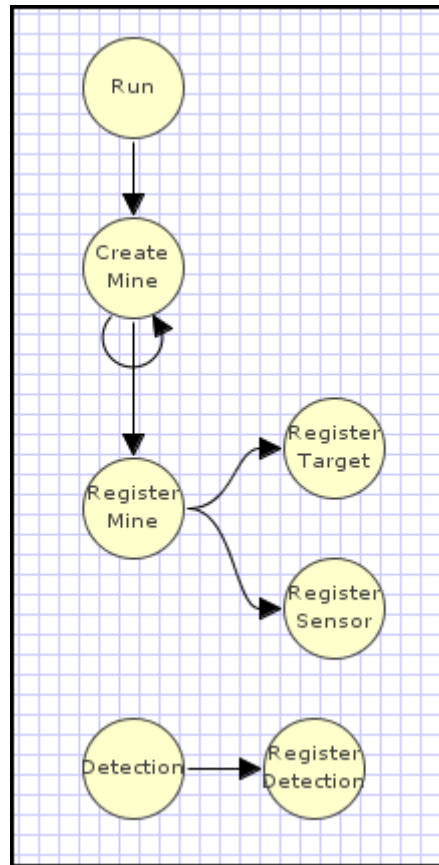
Figure 47.　Minefield event graph.

Figure 47 depicts the Minefield event graph. Its purpose is to create an arbitrary number of mines specified by the numberMines parameter and disperse them in some arbitrary area with some specified distribution. The Run event initializes and resets all state variables then passes control to the Create Mine event. As its name suggests, the Create Mine event creates an amount of DISMover3D mine objects equal to the numberMines parameter. Each mine has a 3D location generated from the three random variate parameters called mineDistributionWidth, mineDistributionLength, and mineDistributionDepth. Each mine is then registered as a mover with the Scenario Manager in the Register Target event. Additionally, a SphereCutterSensor is created for each mine and registered with Scenario Manager in the Register Sensor event. Finally, as all Detection events function, it fires each time a mover object enters within the

76

sensorRange parameter of the sensor created by this event graph. This sequence of events enables the Register Detection to iterate the numberOfDetectionsByMine state variable allowing for collection of statistics on objects detected by mines.

a.   Minefield Parameters

Figure 48.   Minefield parameters as shown in the assembly Event Graph Inspector.

Figure 48 depicts the Event Graph Inspector of the Minefield model. The minefield is simply a number of static movers dispersed over an area in a certain distribution. Each mine is a mover3D object to enable detection capabilities. Table 10 lists and defines each Minefield parameter.

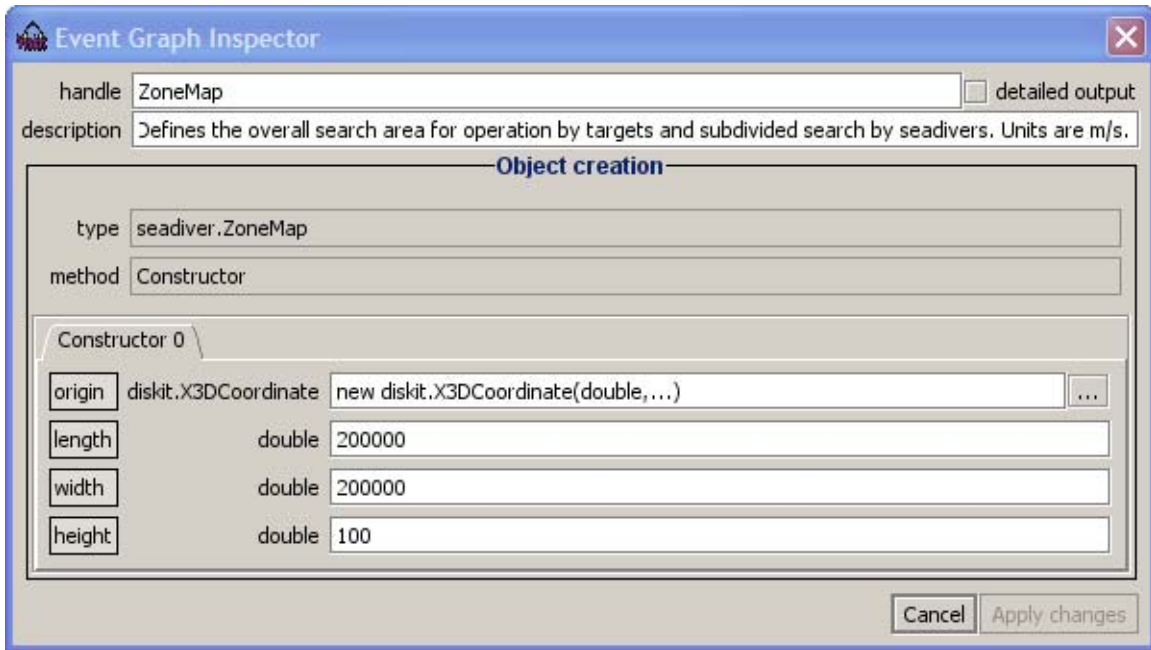| Initialization Parameter	Type, Units	Description
sensorRange	double, m	Determines the range of the sensor for each mine. All mine sensor ranges are equal.
numberMines	int	Determines the number of mine objects that will be created.
firstMoverID	int	Determines the range of individual ID numbers required by DISMover3D objects.
minefieldWidth	double, m	Determines the width (y value) of the minefield.
minefieldLength	double, m	Determines the length (x value) of the minefield.
minefieldDepth	double, m	Determines the depth (z value) of the minefield.
mineDistributionWidth	simkit.random. RandomVariate	Determines the distribution of mines over the width of the minefield.
mineDistributionLength	simkit.random. RandomVariate	Determines the distribution of mines over the length of the minefield.
mineDistributionDepth	simkit.random. RandomVariate	Determines the distribution of mines over the depth of the minefield.

Table 10.   Minefield initialization parameters defined.

b.   *Minefield State Variables*

State Variable	Type	Description
numberOfDetectionsByMine	int	Counter to collect statistics on mover objects detected by mines.

Table 11.   Minefield State Variables.

4.   Surface/Submerged Target Event Graph
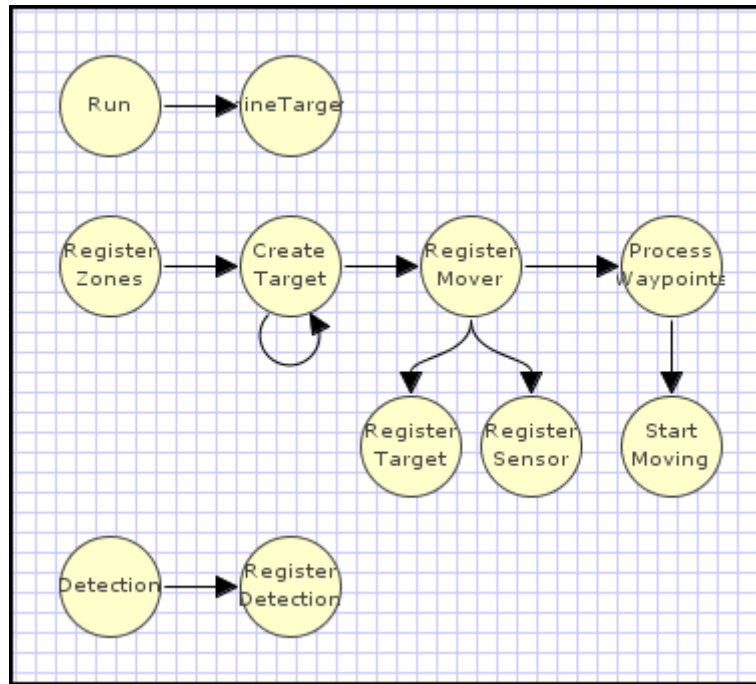
Figure 49.   Surfaced or submerged Target event graph.

The Target event graph is similar to the Seadiver event graph and also has three functional sections: Initialization, Creation of DISMover3D objects, and Detection and Statistics. It simulates a moving surface or submerged vessel such as a destroyer or submarine which also has the capability to detect other mover objects. Currently, Target event graph contains no behavioral logic in the event of a detection event other than logging the detection for statistical calculations.

In the initialization section of Target event graph, the Run event initializes all parameters and state variables then fires the Determine Target Zones event. This event passes the numberOfWaypoints and numberOfTargets parameters to the ZoneMap event graph which uses that information to create zones required for target waypoint generation.

The next section is involved with creation and initialization of the DISMover3D objects. First, the Register Zones event receives the list of zones created by the ZoneMap event graph. A graphical depiction of this zone list is in Figure 45. An instance of TargetWaypointCreator is then created with the zoneList and waypointDistribution in the

signature. Next, an instance of DISMover3D is created for each target mover with an ID beginning at 400 in the Register Mover event. Exactly like the Seadiver event graph, the Register Target and Register Sensor events register the DISMover3D mover and the sensor with the Scenario Manager. Finally, the Process Waypoints event instantiates PathMoverManager with the TargetWaypointCreator created earlier, which generates unique waypoints for each mover and the Start Moving event initiates the movement sequence.

The third section senses and logs detection events by Target movers to allow for collection of statistics. All sensors created in this event graph (one per DISMover3D) are able to be listened to by creation of a SimEventListener connection. This ensures that any appropriate sensor detection events can fire the Detection event in this event graph, which allows for iteration of the numberOfSeadiverDetections state variable and therefore the collection of statistics.

a. Target Parameters

Figure 50.   Target initialization parameters as shown in the assembly Event Graph Inspector.

Figure 50 depicts the Event Graph Inspector of the Target model. The Target model is based on the Seadiver model, minus some functionality not required of targets, and therefore the parameters are similar. Table 12 lists and defines each Target initialization parameters.

Initialization Parameter	Type, Units	Description
speed	double, m/s	Determines the speed of the mover.
numberOfTargets	int	Determines the number of target movers that will be created.
numberOfWaypoints	int	Determines the number of waypoints the target traverses. Zero means the target traverses the area in a straight line.
waypointDistribution	simkit.random. Random Variate	Defines the random distribution of placement of numberOfWaypoints in the area.
inputFileTemplate	String	File name and path used as a template to create AVCL mission files. Note that path is relative to the Viskit Behavior Libraries directory and fwd slashes (/) are required.
outputFileName	String	File name and path to write created AVCL mission files. Note that path is relative to the Viskit Behavior Libraries directory and fwd slashes (/) are required.
operatingDepth	double, m	Determines the nominal operating depth of the Target. Note that this can represent a surfaced or submerged target.
sensorRange	double, m	Determines the range of the SphereCutterSensor attached to each target.

Table 12.   Target initialization parameters defined.

State Variable	Type	Description
zoneList	LinkedList	Collection of zones created and passed by ZoneMap event graph. Used by TargetWaypointCreator to generate waypoints.
active MoverManager	diskit.MoverManager	Mover manager helper that simply holds the current mover manager.
yRandomVariate	Simkit.random. RandomVariate	Random number generation object which generates random numbers in the width direction. Used by seadiver.RandomNumberGenerator class to generate a set of unique random numbers used in generating waypoints.
zRandomVariate	Simkit.random. RandomVariate	Random number generation object which generates random numbers in the height direction. Used by seadiver.RandomNumberGenerator class to generate a set of unique random numbers used in generating waypoints.
random VariateArray	Simkit.random. RandomVariate[3]	Array that holds 3 random numbers passed to TargetWaypointCreator for generation of target waypoints.
random NumberGenerator	seadiver.Random NumberGenerator	Seadiver class which is used to generate unique waypoints for each simulation repetition.
target WaypointGenerator	seadiver.Target WaypointGenerator	Generates unique target waypoints through an area.
numberOf SeadiverDetections	int	Counter to collect statistics on the amount of Seadiver detections.

Table 13.   Target State Variables.

D. SIMULATION DEFINITION: ASSEMBLIES

In Viskit, simulations are constructed as assemblies in the Assembly Panel depicted previously in Figure 31. Assemblies are simply simulation definitions which enable visual representation of the interactions between specific-instance event graphs. Prior to Viskit, the simulation was assembled in a main class which handled the SimEventListener and PropertyChangeListener connections manually. Viskit enables creation of simulations using existing event graphs quickly and easily with drag and drop simplicity.

Event graphs are connected together via SimEventListener connections that allow the listening event graph knowledge of specific events in the listened to event graph. This allows for information to flow between event graphs, enabling behaviors such as one entity reacting to another entity, and is accomplished by simply drawing a line between one event to the next with a SimEventListener connector selected. Similarly, assemblies allow for the easy collection of statistics by connecting specific event graphs and PropertyChangeListeners.

Despite the simplicity exposed in assemblies for creating SimEventListener and PropertyChangeListener connections, a detailed knowledge of the event graphs used is required. In fact, except for the simplest event graphs, any simulation requiring advanced behaviors demands that the event graphs be created with the intent of being integrated into a specific assembly from the beginning. For example, the Seadiver simulation is designed around reusing proven diskit components such as Scenario Manager and DISMover3D. In that context, the Seadiver and Target event graphs must have Register Target and Register Sensor events, and a SimEventListener connection must be established from Scenario Manager to the other event graphs or the simulation will not start. In short, the conceptual design of behavior models, the creation of event graphs based on those behaviors, and the integration of those event graphs into an assembly must be conducted with each in mind.

1. Minefield Search Assembly
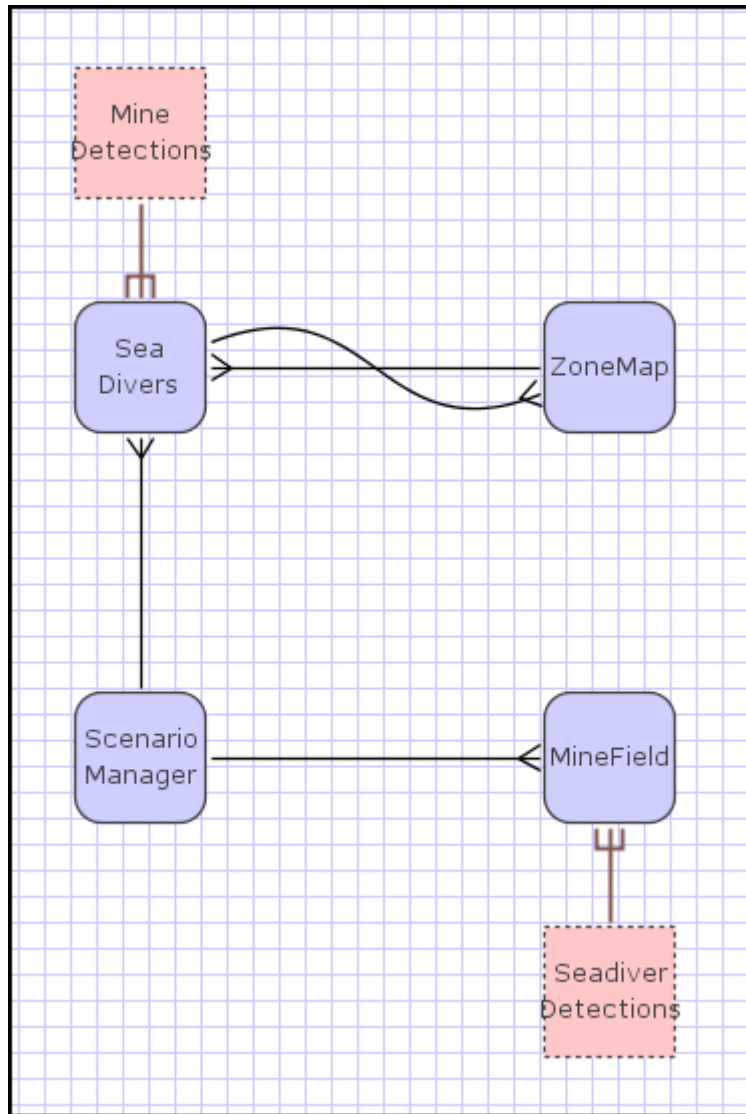
Figure 51.   Minefield Search Assembly.

The Minefield Search Assembly consists of four event graphs and two PropertyChangeListeners as shown in Figure 51. The mediator between movers and sensors, Scenario Manager, listens to the Register Target and Register Sensor events in both the Seadiver and Minefield event graphs via a SimEventListener connection. The Seadiver and ZoneMap event graphs listen to each other with ZoneMap listening for the Create Map event in Seadiver while Seadiver listens for the Register Map event in ZoneMap. As discussed earlier, these connections are required to sectionalize the entire operating area into the appropriate amount of zones, and to assign each Seadiver to an

area. The Minefield event graph is independent of the ZoneMap event graph and no limitations are set for the Minefield size, which enables the Minefield to be arbitrarily larger or smaller than the Seadiver operating area.

The PropertyChangeListeners associated with the Minefield Search assembly are Mine Detections and Seadiver Detections which are of type SimpleStatsTally. SimpleStatsTally simply keeps a running total of property changes and calculates statistical values such as minimum, maximum, mean, variance and standard deviation at the end of a repetition. They are connected via PropertyChangeListener connections to the event graphs that expose the appropriate state variables, namely mineDetections and numberOfDetectionsByMine respectively. Establishing PropertyChangeListeners in this manner allows for Viskit to automatically display repetition and summary statistics which are displayed in the Assembly Run panel. Further analysis support is also provided by the Analyst Report tabbed panes.

2. Barrier Search Assembly
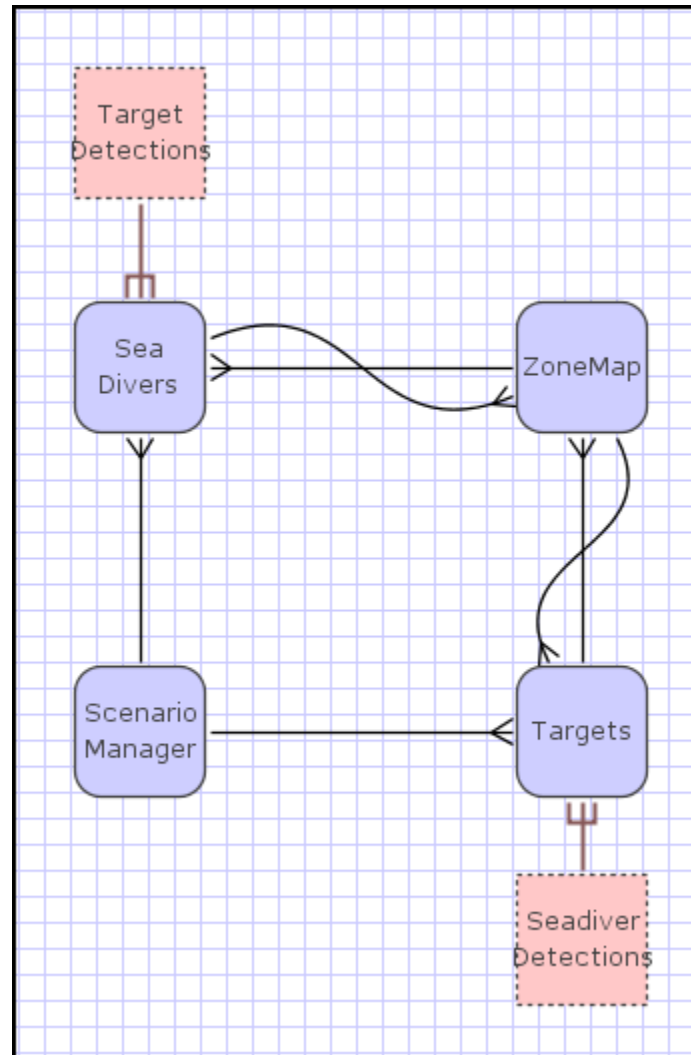
Figure 52.   Barrier Search Assembly.

The Barrier Search Assembly consists of four event graphs and two PropertyChangeListeners as shown in Figure 52. The mediator between movers and sensors, Scenario Manager, listens to the Register Target and Register Sensor events in both the Seadiver and Minefield event graphs via a SimEventListener connection. Seadivers and ZoneMap event graphs listen to each other with ZoneMap listening for the Create Map event in Seadiver while Seadiver listens for the Register Map event in ZoneMap. As discussed earlier, these connections are required to sectionalize the entire operating area into the appropriate amount of zones and to expose Seadiver to an area

without requiring prior knowledge of it in the event graph. Similarly, the Target and ZoneMap event graphs listen to each other with ZoneMap listening to Determine Target Zones event and Target listening to the Register Zones event.

The PropertyChangeListeners associated with the Barrier Search assembly are Target Detections and Seadiver Detections which are of type SimpleStatsTally. SimpleStatsTally simply keeps a running total of property changes and calculates statistical values such as minimum, maximum, mean, variance and standard deviation at the end of a repetition. They are connected via PropertyChangeListener connections to the event graphs that expose the appropriate state variables, namely mineDetections and numberOfSeadiverDetections respectively. Establishing PropertyChangeListeners in this manner allows for Viskit to automatically display repetition and summary statistics which are displayed in the Assembly Run panel.

E.   MISSION VALIDATION WITH AUVW

This thesis leverages the Autonomous Unmanned Vehicle Workbench (AUVW), a virtual environment simulation program detailed in Chapter II, to validate mission generation and entity behavior for conducting missions. When designing complex simulations in a programming environment, it becomes increasingly difficult to ensure moving entities are operating as desired. This is especially true of this simulation because many moving entities are required to work in concert over a large area simultaneously.

The AUVW is used in this case to visually verify that each moving entity is correctly positioned in its assigned operating zone and that waypoint generation for that entity was successful. Specifically the Viskit classes produce AVCL mission scripts for the respective targets and Seadiver vehicles to follow. Improper movement of entities leads to difficult or faulty analysis of statistical results that can be difficult to diagnose. Figure 53 demonstrates visual validation of a notional simulation consisting of ten Seadiver entities and two Target entities.
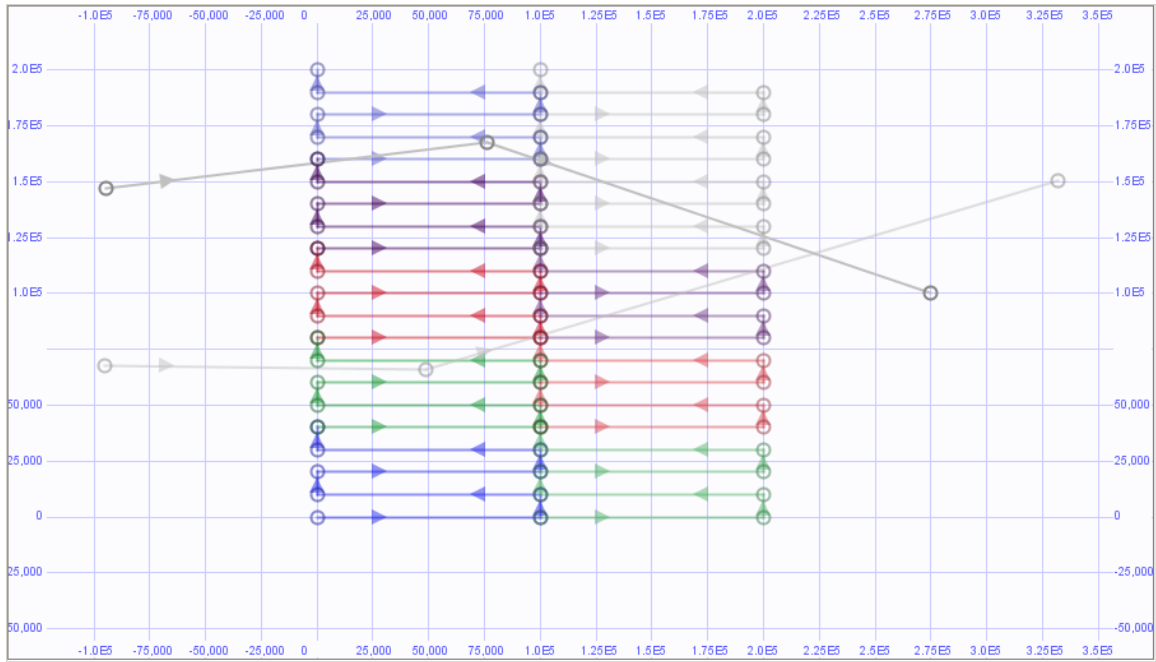
Figure 53.　Visual path validation with AUVW, displaying 10 Seadiver entities searching and 2 Target entities transiting.

The Seadiver simulation makes several assumptions about the physical operating characteristics of the Seadiver entity for simplification that could invalidate the statistical results if not determined to have negligible effect of the simulation. For example, it is assumed that Seadiver vector changes are instantaneous, which is obviously incorrect since every moving entity has some turning radius. On a small-enough scale as with search legs within a short distance, this assumption becomes non-negligible. Validation using AUVW to run the AVCL missions can determine proper production of missions.

In AUVW, the Seadiver UUV has been physically and dynamically modeled. This enables not only validation of generated search paths, but validation that the Seadiver UUV can physically navigate this path with realistic physical characteristics, such as diving or rising to generate forward propulsion, or the ability to navigate a turn successfully. By running Seadiver AVCL missions in The AUVW generated by the Seadiver simulation and observing correct behavior, validation of Seadiver operating characteristics in the context of mission generated by the Seadiver simulation is established.

F. SUMMARY

This chapter describes in detail the logic and methods employed including required assumptions to create event graph models and simulation assemblies for this research. Beginning with Section B, the assumptions and logic behind them are detailed to produce a fairly abstract model. Section C detailed how entity behaviors are captured in this research during the creation of the four event graphs that compose this thesis. The event graphs represent three moving entities (Seadiver, Target, and Minefield) and one area knowledge object (ZoneMap). Section D detailed how the aforementioned event graphs along with Scenario Manager were integrated into the functional simulations. Section D also explains how this thesis constructs two functional assemblies and therefore two simulations which mimic the ability of the Seadiver to conduct relevant military missions. Finally, Section E describes how AUVW is employed to validate simulation results and why that is required in this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

VII.   SIMULATION RESULTS

This simulation not intended to be analytically rigorous. That is, the statistics generated do not represent that the Seadiver can perform these missions, only that a DES can be made with Viskit to model the Seadiver UUV and its hypothetical missions. This simulation can however be used as a framework for follow-on simulations that are analytically rigorous.

Simulation results and analysis are contained in Appendices A and B for the Barrier Search and Minefield Search missions respectively.

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. CONCLUSIONS AND RECOMMENDATIONS

A.   CONCLUSIONS

1.   Discrete Event Model Creation

This research has created DES event graph models of various moving entities defined by specific behaviors. These models, Seadiver, Target, Minefield, and ZoneMap are scalable, repeatable, re-useable, and modifiable. This thesis has explained in detail the process of creating Discrete Event simulations from the start by creation of entity definitions via event graphs to execution of simulations and result generation. Using this thesis as a guide, new models can be created and implemented with the programs and APIs outlined. Additionally, the models and supporting classes can be used as is or modified and used in new simulations.

2.   Mission-Structured Simulations

The event graphs and supporting Java classes created during this research are used to create two simulations structured around the unique capabilities of the Seadiver UUV. These missions are the Barrier Search and the Minefield Search, both implemented over large areas with numerous Seadiver UUVs. These simulations can be used as a repeatable framework to analyze in detail the capabilities of the Seadiver UUV in performance of these missions or as a platform for creation of additional simulations based on missions.

3.   Simulation Validation with AUVW

Validation of simulation mission structure and dynamic constraints is required and demonstrated for this research. Validation is accomplished in AUVW through the implementation of missions that are automatically generated during Viskit simulation initialization by a Java class designed to produce mission files in AVCL format from a Discrete Event simulation. Individual and small group can be validated coherently, but large scale missions incorporating many entities (threshold: 20) are not feasible in AUVW at this time due to limited computational resources of recent desktop computers. This limitation can be worked around satisfactorily by using multiple computers, since all

results are networked and sharable. Further AUVW software development to improve threading and resource allocation is expected to provide significant improvements in computational efficiency and mission capacity.

B. RECOMMENDATIONS FOR FUTURE WORK

1. Real-World Classified Study

This thesis provides the framework and approach for conducting a real-world classified study of Seadiver performing difficult missions. The two mission simulations created in this thesis are an exemplar to other unique and currently infeasible UUV missions that can be constructed in Viskit based on this research. These missions include a Mobile Minefield or Large-Area Moving Underwater Communication Network. To enable a real-world study, more research is required to determine Seadiver physical capabilities and specifics of available sensor capabilities.

2. Increased Sensor Fidelity

To generate realistically accurate results from these mission simulations, there is a need to expand the Diskit sensor library both in capability and fidelity. Sensor objects need to be created for all expected sensor packages on UUVs performing these missions to evaluate the most appropriate sensor or combination of sensors. Additionally, these sensor objects need to accurately reflect the capabilities of the modeled sensor to provide for realistic mission results.

3. Finish Design, Construction and Validation Testing of Seadiver

The Seadiver UUV has not been physically completed nor tested in a water environment. Therefore it is difficult to accurately predict the hydrodynamic and behavioral characteristics of Seadiver. These characteristics include turning radius, speed through the water, and rate of change of depth, and are required to create simulations of higher fidelity. Additionally, the process of construction and testing will validate some assumptions made in this thesis such as Seadiver's low cost of construction and speed.

4. Implement Advanced Search Techniques

Currently, this thesis implements a lawn-mower search pattern for each mission type. This is adequate for a minefield search since 100 percent of the area must be searched, but is inefficient for a barrier search. There is a need to implement advanced search techniques such as the A-Star search path optimization or other heuristic search methods to reduce these inefficiencies. Additionally, providing more search options allows the researcher to test optimum combinations of parameters applicable to search patters.

5. Implement Inheritance for Viskit Event Graph Models

Inheritance is used extensively in this research to simplify event graphs based on behaviors. This allowed the event graphs of entities to only define the behaviors unique to each. Currently, Viskit event graphs cannot extend other event graphs because of the nature of the XML format in which they are saved. Therefore, all parent classes of the event graph models that were created in this thesis had to be defined as Java code. To allow for easier implementation of inheritance, there is a need for native Viskit event graphs to extend each other.

6. Programmatically Load Viskit Assemblies for Large Simulations

This thesis employed special techniques to create multiple moving entities in a single event graph. This created difficulties during the simulation phase due to the method of repetition implementation in Simkit, necessitating special programming measures to overcome. It is recommended that many entities not be created on a single event graph to allow for focus to be maintained on entity behavior alone. Instead, a programmatic method for loading multiple (tens to hundreds) entities into assemblies should be researched to separate entity mechanics and behaviors when implementing large arrays or lists of identical entities in a given assembly.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS REPORT IS: **UNCLASSIFIED**

Barrier Search Simulation

Analyst: **LT John M. Seguin**
Analysis date: **3/19/07 2:17 PM**

Executive Summary

A novel UUV is currently being designed that is projected to support significantly greater endurance and range characteristics. This UUV is called Seadiver and is being designed by Institute of Engineering Science of Toulon, France with support from Naval Postgraduate School. It is a low-cost glider UUV which generates propulsion not with propellers or jet pumps, but rather by controlling its buoyancy. This method of propulsion is quite efficient and maybe capable of autonomous operation up to 30 days with a range of around 700 nautical miles. A UUV with such endurance and range exposes military missions previously impractical for UUVs especially when used in concert as an array of many UUVs

This simulation models the ability of Seadiver UUV to perform the Barrier Search mission. The Barrier Search mission's purpose is to detect hostile contacts moving across the barrier. The barrier is comprised of 100 searching Seadiver UUVs spread symmetrically across the area. As the hostile contacts traverse from one side of the area to the other, they are detected by Seadiver UUVs if they come within the sensor range of the UUV. All unique detections are logged and statistical results are generated.

Simulation Location

Description of Location Features. This mission takes place in a generic littoral ocean area 200 km long, 200 km wide, and 100 m deep.

Post-Experiment Analysis of Significant Location Features. This was a generic area. The area can be modified to meet the needs of the analyst.

Assembly Configuration for Viskit Simulation

Assembly Design Considerations. This assembly is designed around the Barrier Search mission. The ZoneMap node takes the total area dimensions and creates individual operating zones for each Seadiver UUV, and waypoint zones for the hostile contacts (Target entities). This information is passed to the SeaDivers and Targets nodes respectively through simEventListener connections. Scenario Manager controls and manages movement and detection between all entities. There are two statistics collecting nodes called TargetDetections and SeadiverDetections connected to the applicable entity nodes containing the State Variables with Property Change Listeners.

Post-Experiment Analysis of Simulation Assembly Design. Simulation works as designed, but behavior implementation was problematic due to creation of many moving entities in a single node. A possible solution could be to programmatically generate the assembly and have one moving entity per node.

Summary of Simulation Entities

Entity Name	Behavior Definition
SeaDivers	seadiver.SeaDiver
ZoneMap	seadiver.ZoneMap
Targets	seadiver.Target
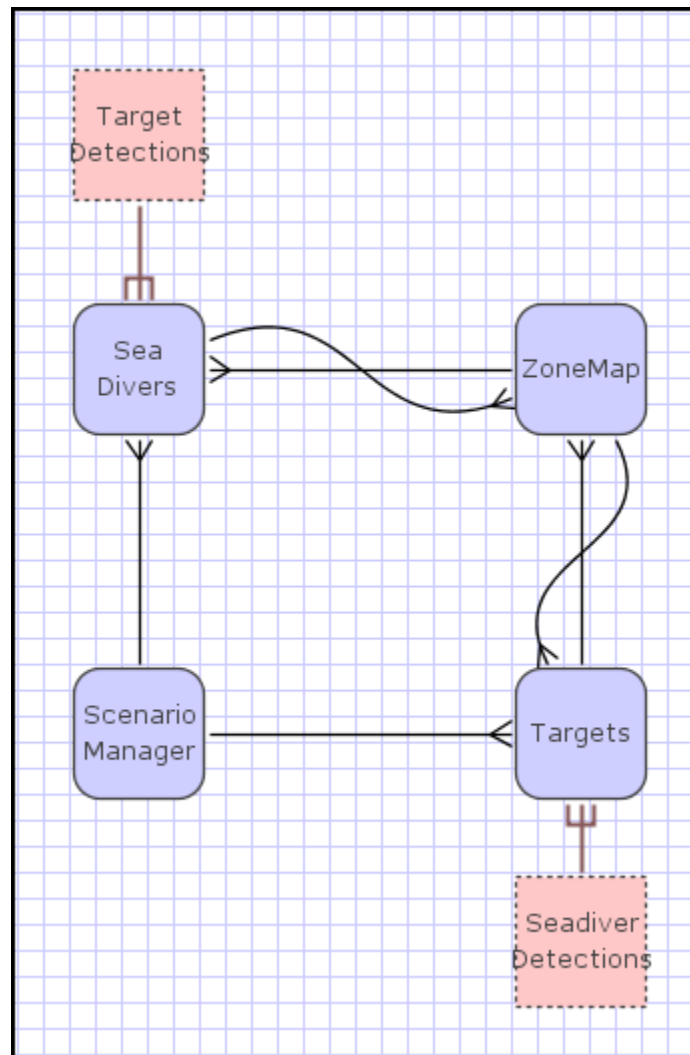
Entity Parameters

Entity Parameters Overview. Entity parameters are initialization values used to define new event graphs. These values are pulled directly from the assembly.

Behavior Definitions

Description of Behavior Design. Seadiver: Seadiver is modeled after the SeaDiver glider UUV. Its main behavior is to conduct a search in a Lawn-mower pattern over an area. In the Barrier Search mission, the entities search their respective areas for other moving entities (Targets). Upon detection, the UUV immediately takes a GPS fix (simulated) then continues the search indefinitely.

Target: Target is modeled after a moving submerged or surfaced contact that is traversing an area patrolled by Seadiver UUV. Targets have the ability to detect Seadiver UUVs, but take no action upon detection.

Post-Experiment Analysis of Entity Behaviors. Possible future behavior would be to incorporate behavior for Targets upon detection of Seadiver UUVs.

Statistical Results

Description of Expected Results. This simulation initiated 50 repetitions displayed below. This is used as an exemplar to indicate the correct application of detection in a DES, and the ability to produce representative statistics from that behavior. SeadiverDetections statistic indicates the amount of Seadivers that were detected by Targets. TargetDetections statistic indicates that all Targets were detected each repetition.

Analysis of Experimental Results. As an exemplar, no analysis was performed. Other potential useful statistics would be time to detect all targets or the amount of detections per target.

Replication Report
Entity: Target
Property: SeadiverDetections
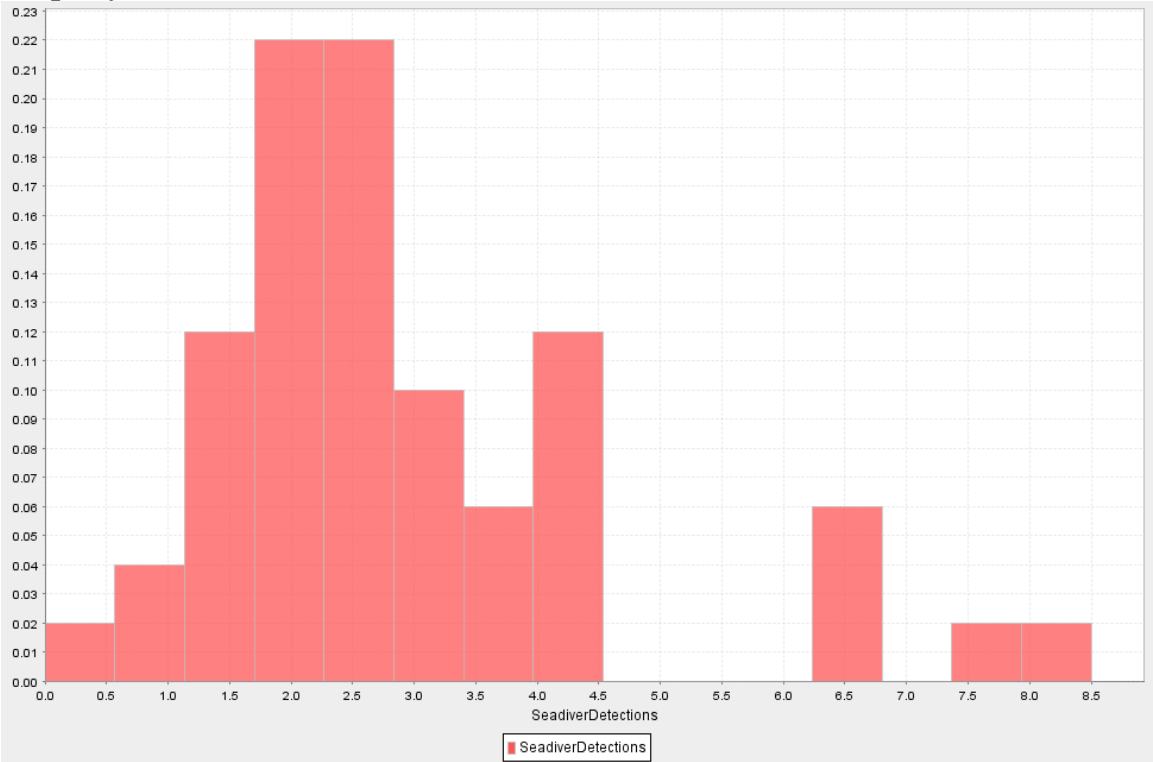
Run#	Count	Min	Max	Mean	StdDev	Variance
1	14.000	0.000	13.000	6.500	4.183	17.500
2	4.000	0.000	3.000	1.500	1.291	1.667
3	10.000	0.000	9.000	4.500	3.028	9.167
4	6.000	0.000	5.000	2.500	1.871	3.500
5	6.000	0.000	5.000	2.500	1.871	3.500
6	4.000	0.000	3.000	1.500	1.291	1.667
7	5.000	0.000	4.000	2.000	1.581	2.500
8	8.000	0.000	7.000	3.500	2.449	6.000
9	4.000	0.000	3.000	1.500	1.291	1.667
10	5.000	0.000	4.000	2.000	1.581	2.500
11	4.000	0.000	3.000	1.500	1.291	1.667
12	4.000	0.000	3.000	1.500	1.291	1.667
13	14.000	0.000	13.000	6.500	4.183	17.500
14	7.000	0.000	6.000	3.000	2.160	4.667
15	6.000	0.000	5.000	2.500	1.871	3.500
16	16.000	0.000	15.000	7.500	4.761	22.667

Run#	Count	Min	Max	Mean	StdDev	Variance
17	3.000	0.000	2.000	1.000	1.000	1.000
18	5.000	0.000	4.000	2.000	1.581	2.500
19	5.000	0.000	4.000	2.000	1.581	2.500
20	9.000	0.000	8.000	4.000	2.739	7.500
21	18.000	0.000	17.000	8.500	5.339	28.500
22	5.000	0.000	4.000	2.000	1.581	2.500
23	8.000	0.000	7.000	3.500	2.449	6.000
24	6.000	0.000	5.000	2.500	1.871	3.500
25	5.000	0.000	4.000	2.000	1.581	2.500
26	4.000	0.000	3.000	1.500	1.291	1.667
27	5.000	0.000	4.000	2.000	1.581	2.500
28	5.000	0.000	4.000	2.000	1.581	2.500
29	10.000	0.000	9.000	4.500	3.028	9.167
30	1.000	0.000	0.000	0.000	0.000	0.000
31	6.000	0.000	5.000	2.500	1.871	3.500
32	6.000	0.000	5.000	2.500	1.871	3.500
33	7.000	0.000	6.000	3.000	2.160	4.667
34	7.000	0.000	6.000	3.000	2.160	4.667
35	6.000	0.000	5.000	2.500	1.871	3.500
36	9.000	0.000	8.000	4.000	2.739	7.500
37	5.000	0.000	4.000	2.000	1.581	2.500
38	6.000	0.000	5.000	2.500	1.871	3.500
39	9.000	0.000	8.000	4.000	2.739	7.500
40	7.000	0.000	6.000	3.000	2.160	4.667
41	3.000	0.000	2.000	1.000	1.000	1.000
42	6.000	0.000	5.000	2.500	1.871	3.500
43	7.000	0.000	6.000	3.000	2.160	4.667
44	6.000	0.000	5.000	2.500	1.871	3.500
45	5.000	0.000	4.000	2.000	1.581	2.500
46	5.000	0.000	4.000	2.000	1.581	2.500
47	8.000	0.000	7.000	3.500	2.449	6.000
48	6.000	0.000	5.000	2.500	1.871	3.500
49	9.000	0.000	8.000	4.000	2.739	7.500
50	14.000	0.000	13.000	6.500	4.183	17.500

Replication Report
Entity: Seadivers
Property: TargetDetections
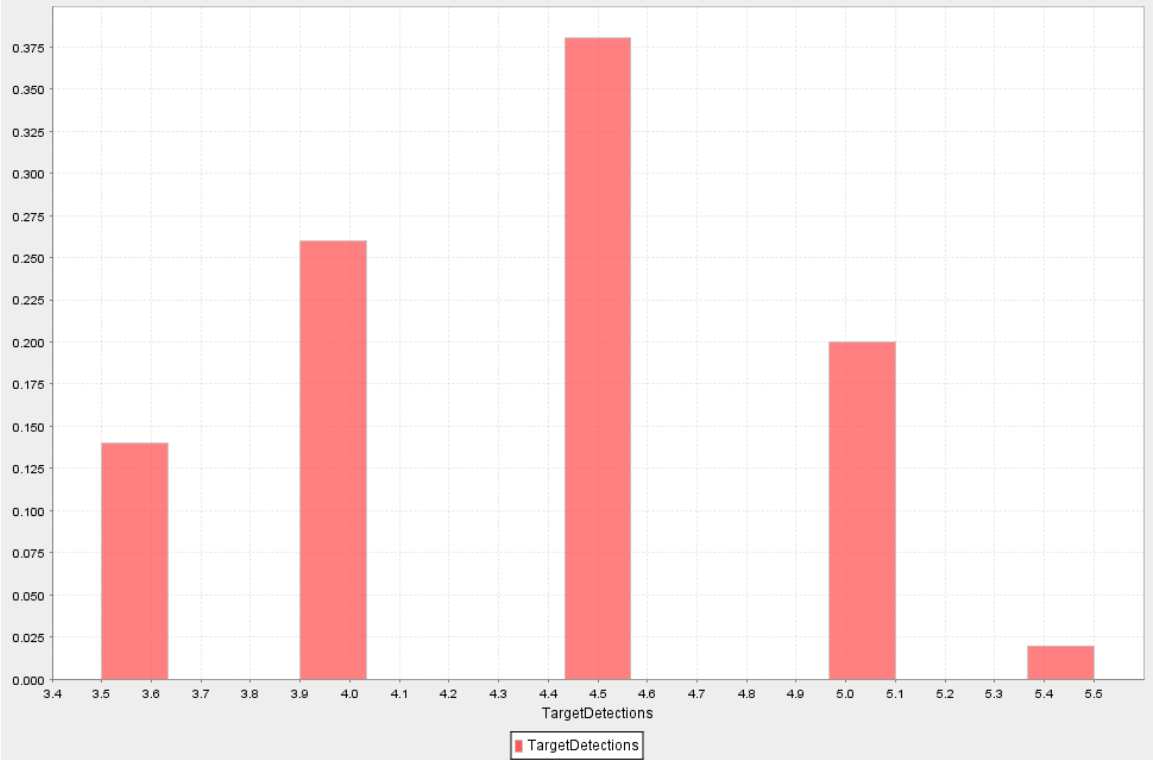
Run#	Count	Min	Max	Mean	StdDev	Variance
1	10.000	0.000	9.000	4.500	3.028	9.167
2	12.000	0.000	11.000	5.500	3.606	13.000
3	11.000	0.000	10.000	5.000	3.317	11.000
4	9.000	0.000	8.000	4.000	2.739	7.500
5	8.000	0.000	7.000	3.500	2.449	6.000
6	11.000	0.000	10.000	5.000	3.317	11.000
7	10.000	0.000	9.000	4.500	3.028	9.167
8	11.000	0.000	10.000	5.000	3.317	11.000
9	9.000	0.000	8.000	4.000	2.739	7.500
10	8.000	0.000	7.000	3.500	2.449	6.000
11	9.000	0.000	8.000	4.000	2.739	7.500
12	10.000	0.000	9.000	4.500	3.028	9.167
13	10.000	0.000	9.000	4.500	3.028	9.167
14	10.000	0.000	9.000	4.500	3.028	9.167
15	11.000	0.000	10.000	5.000	3.317	11.000
16	10.000	0.000	9.000	4.500	3.028	9.167

Run#	Count	Min	Max	Mean	StdDev	Variance
17	9.000	0.000	8.000	4.000	2.739	7.500
18	11.000	0.000	10.000	5.000	3.317	11.000
19	9.000	0.000	8.000	4.000	2.739	7.500
20	10.000	0.000	9.000	4.500	3.028	9.167
21	10.000	0.000	9.000	4.500	3.028	9.167
22	10.000	0.000	9.000	4.500	3.028	9.167
23	9.000	0.000	8.000	4.000	2.739	7.500
24	10.000	0.000	9.000	4.500	3.028	9.167
25	10.000	0.000	9.000	4.500	3.028	9.167
26	9.000	0.000	8.000	4.000	2.739	7.500
27	8.000	0.000	7.000	3.500	2.449	6.000
28	8.000	0.000	7.000	3.500	2.449	6.000
29	11.000	0.000	10.000	5.000	3.317	11.000
30	10.000	0.000	9.000	4.500	3.028	9.167
31	11.000	0.000	10.000	5.000	3.317	11.000
32	10.000	0.000	9.000	4.500	3.028	9.167
33	10.000	0.000	9.000	4.500	3.028	9.167
34	9.000	0.000	8.000	4.000	2.739	7.500
35	9.000	0.000	8.000	4.000	2.739	7.500
36	11.000	0.000	10.000	5.000	3.317	11.000
37	10.000	0.000	9.000	4.500	3.028	9.167
38	8.000	0.000	7.000	3.500	2.449	6.000
39	10.000	0.000	9.000	4.500	3.028	9.167
40	10.000	0.000	9.000	4.500	3.028	9.167
41	10.000	0.000	9.000	4.500	3.028	9.167
42	11.000	0.000	10.000	5.000	3.317	11.000
43	9.000	0.000	8.000	4.000	2.739	7.500
44	8.000	0.000	7.000	3.500	2.449	6.000
45	10.000	0.000	9.000	4.500	3.028	9.167
46	11.000	0.000	10.000	5.000	3.317	11.000
47	9.000	0.000	8.000	4.000	2.739	7.500
48	9.000	0.000	8.000	4.000	2.739	7.500
49	9.000	0.000	8.000	4.000	2.739	7.500
50	8.000	0.000	7.000	3.500	2.449	6.000

Summary Report

Entity	Property	Count	Min	Max	Mean	StdDev	Variance
Seadiver	TargetDetections	50.000	3.500	5.500	4.350	0.508	0.258
Target	SeadiverDetections	50.000	0.000	8.500	2.930	1.693	2.867

Conclusions and Recommendations

Conclusions. This simulation was a good exemplar for testing Barrier Search missions for the SeaDiver UUV. For specific conclusions, see Chapter 8.

Recommendations for Future Work. As an exemplar simulation, the behaviors and capabilities of moving entities are very general. More work is required refining the model definitions and search patterns to generate relevant statistics. See Chapter 8 for other specific recommendations for future work.

APPENDIX B.

Minefield Search

Analyst: **LT John M. Seguin**
Analysis date: **3/20/07 12:45 PM**

Executive Summary

Analyst Executive Summary. A novel UUV is currently being designed that is projected to support significantly greater endurance and range characteristics. This UUV is called Seadiver and is being designed by Institute of Engineering Science of Toulon, France with support from Naval Postgraduate School. It is a low-cost glider UUV which generates propulsion not with propellers or jet pumps, but rather by controlling its buoyancy. This method of propulsion is quite efficient and maybe capable of autonomous operation up to 30 days with a range of around 700 nautical miles. A UUV with such endurance and range exposes military missions previously impractical for UUVs especially when used in concert as an array of many UUVs.

This simulation models the ability of Seadiver UUV to perform the Minefield Search mission. The Minefield Search mission's purpose is to detect all mines in a minefield, then report all locations of mines for later removal. The search is comprised of 100 Seadiver UUVs spread symmetrically across the minefield. The minefield is comprised of 200 mines randomly distributed across the minefield. The Seadiver UUVs conduct the search using a lawn-mower search pattern. All unique mine detections are logged and statistical results are generated.

Simulation Location

Description of Location Features. This mission takes place in a generic littoral ocean area 200 km long, 200 km wide, and 100 m deep.

Post-Experiment Analysis of Significant Location Features. This was a generic area. The area can be modified to meet the needs of the analyst.

Assembly Configuration for Viskit Simulation

Assembly Design Considerations. This assembly is designed around the Minefield Search mission. The ZoneMap node takes the total area dimensions and creates individual operating zones for each Seadiver UUV. This information is passed to the SeaDivers node through simEventListener connections. Scenario Manager controls and manages movement and detection between all entities. There are two statistics collecting nodes called MineDetections and DetectionsByMines connected to the applicable entity node containing the State Variable with Property Change Listeners.

Post-Experiment Analysis of Simulation Assembly Design. Simulation works as designed, but behavior implementation was problematic due to creation of many moving entities in a single node. A possible solution could be to programmatically generate the assembly and have one moving entity per node.

Summary of Simulation Entities

Entity Name	Behavior Definition
ZoneMap	seadiver.ZoneMap
SeaDivers	seadiver.SeaDiver
MineField	seadiver.MineField
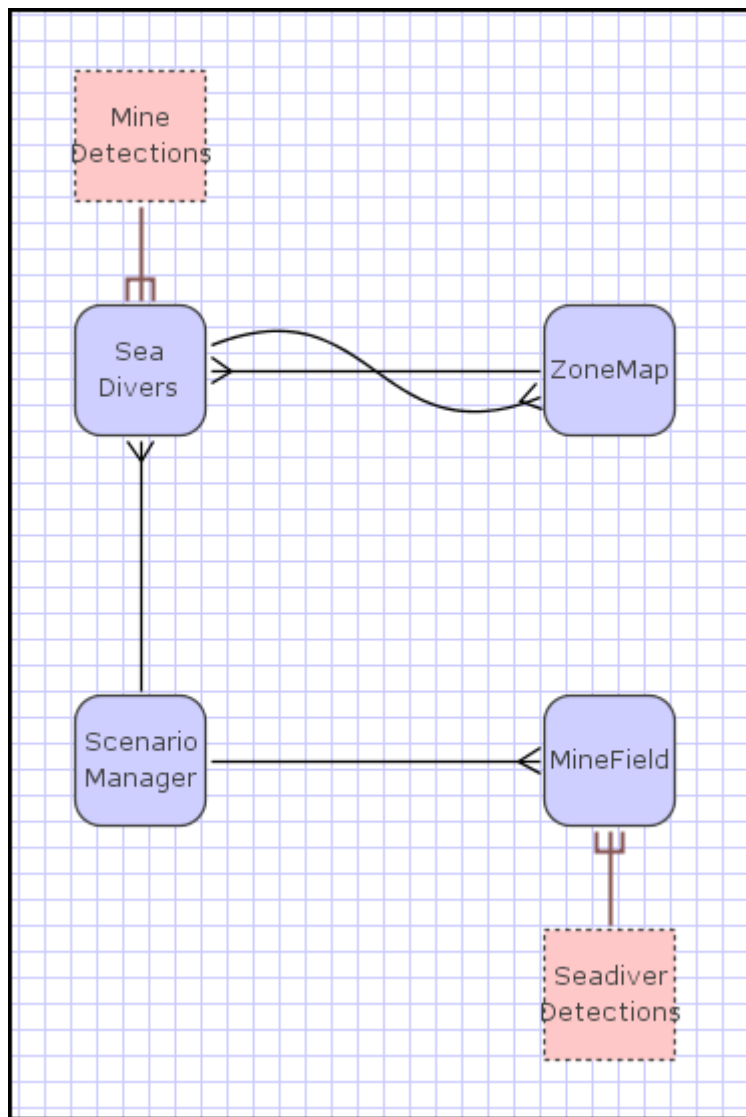
Entity Parameters

Entity Parameters Overview. Entity parameters are initialization values used to define new event graphs. These values are pulled directly from the assembly.

Behavior Definitions

Description of Behavior Design. Seadiver: Seadiver is modeled after the SeaDiver glider UUV. Its main behavior is to conduct a search in a lawn-mower pattern over its section of the minefield. In the Minefield Search mission, the entities search their respective areas for mines. Upon detection, the UUV immediately takes a GPS fix (simulated) then continues the search indefinitely.

Minefield: Mines are stationary objects capable of detecting other moving entities. Mines have the ability to detect Seadiver UUVs, but take no action upon detection.

Post-Experiment Analysis of Entity Behaviors. Possible future behavior would be to incorporate behavior for mines upon detection of Seadiver UUVs such as explode.

Statistical Results

Description of Expected Results. This simulation initiated 50 repetitions displayed below. This is used as an exemplar to indicate the correct application of detection in a DES, and the ability to produce representative statistics from that behavior. MineDetections statistic indicates the amount of mines that were detected by Seadivers. DetectionsByMines statistic indicates the number of Seadivers that were detected each repetition by mines.

Analysis of Experimental Results. As an exemplar, no analysis was performed. Other potential useful statistics would be time to detect all mines.

Replication Report
Entity: Seadivers
Property: MineDetections
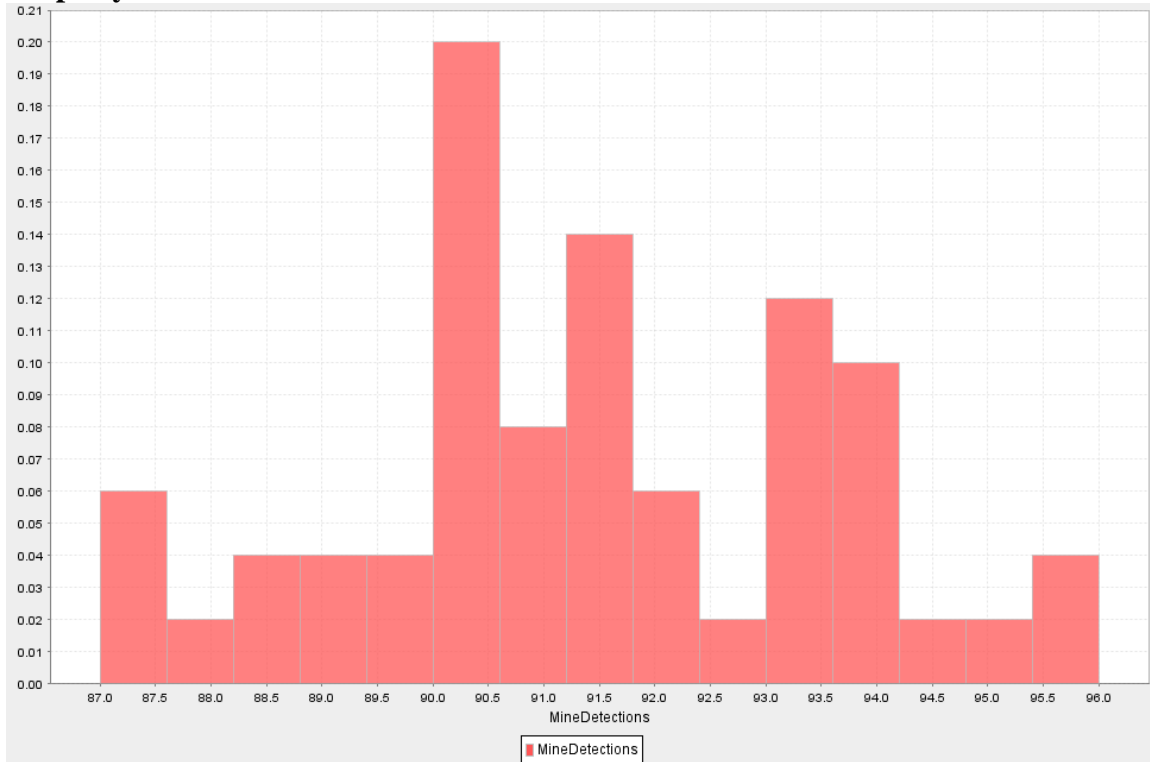
Run#	Count	Min	Max	Mean	StdDev	Variance
1	189.000	0.000	188.000	94.000	54.704	2992.500
2	184.000	0.000	183.000	91.500	53.260	2836.667
3	177.000	0.000	176.000	88.000	51.240	2625.500
4	189.000	0.000	188.000	94.000	54.704	2992.500
5	184.000	0.000	183.000	91.500	53.260	2836.667
6	190.000	0.000	189.000	94.500	54.992	3024.167
7	179.000	0.000	178.000	89.000	51.817	2685.000
8	187.000	0.000	186.000	93.000	54.126	2929.667
9	189.000	0.000	188.000	94.000	54.704	2992.500
10	188.000	0.000	187.000	93.500	54.415	2961.000
11	180.000	0.000	179.000	89.500	52.106	2715.000
12	181.000	0.000	180.000	90.000	52.394	2745.167
13	188.000	0.000	187.000	93.500	54.415	2961.000
14	187.000	0.000	186.000	93.000	54.126	2929.667
15	193.000	0.000	192.000	96.000	55.858	3120.167
16	182.000	0.000	181.000	90.500	52.683	2775.500

Run#	Count	Min	Max	Mean	StdDev	Variance
17	186.000	0.000	185.000	92.500	53.838	2898.500
18	183.000	0.000	182.000	91.000	52.972	2806.000
19	181.000	0.000	180.000	90.000	52.394	2745.167
20	181.000	0.000	180.000	90.000	52.394	2745.167
21	182.000	0.000	181.000	90.500	52.683	2775.500
22	178.000	0.000	177.000	88.500	51.528	2655.167
23	187.000	0.000	186.000	93.000	54.126	2929.667
24	183.000	0.000	182.000	91.000	52.972	2806.000
25	183.000	0.000	182.000	91.000	52.972	2806.000
26	181.000	0.000	180.000	90.000	52.394	2745.167
27	183.000	0.000	182.000	91.000	52.972	2806.000
28	184.000	0.000	183.000	91.500	53.260	2836.667
29	184.000	0.000	183.000	91.500	53.260	2836.667
30	188.000	0.000	187.000	93.500	54.415	2961.000
31	185.000	0.000	184.000	92.000	53.549	2867.500
32	181.000	0.000	180.000	90.000	52.394	2745.167
33	189.000	0.000	188.000	94.000	54.704	2992.500
34	179.000	0.000	178.000	89.000	51.817	2685.000
35	185.000	0.000	184.000	92.000	53.549	2867.500
36	181.000	0.000	180.000	90.000	52.394	2745.167
37	184.000	0.000	183.000	91.500	53.260	2836.667
38	189.000	0.000	188.000	94.000	54.704	2992.500
39	176.000	0.000	175.000	87.500	50.951	2596.000
40	178.000	0.000	177.000	88.500	51.528	2655.167
41	182.000	0.000	181.000	90.500	52.683	2775.500
42	192.000	0.000	191.000	95.500	55.570	3088.000
43	175.000	0.000	174.000	87.000	50.662	2566.667
44	182.000	0.000	181.000	90.500	52.683	2775.500
45	184.000	0.000	183.000	91.500	53.260	2836.667
46	191.000	0.000	190.000	95.000	55.281	3056.000
47	180.000	0.000	179.000	89.500	52.106	2715.000
48	175.000	0.000	174.000	87.000	50.662	2566.667
49	184.000	0.000	183.000	91.500	53.260	2836.667
50	185.000	0.000	184.000	92.000	53.549	2867.500

Replication Report
Entity: Minefield
Property: DetectionsByMine
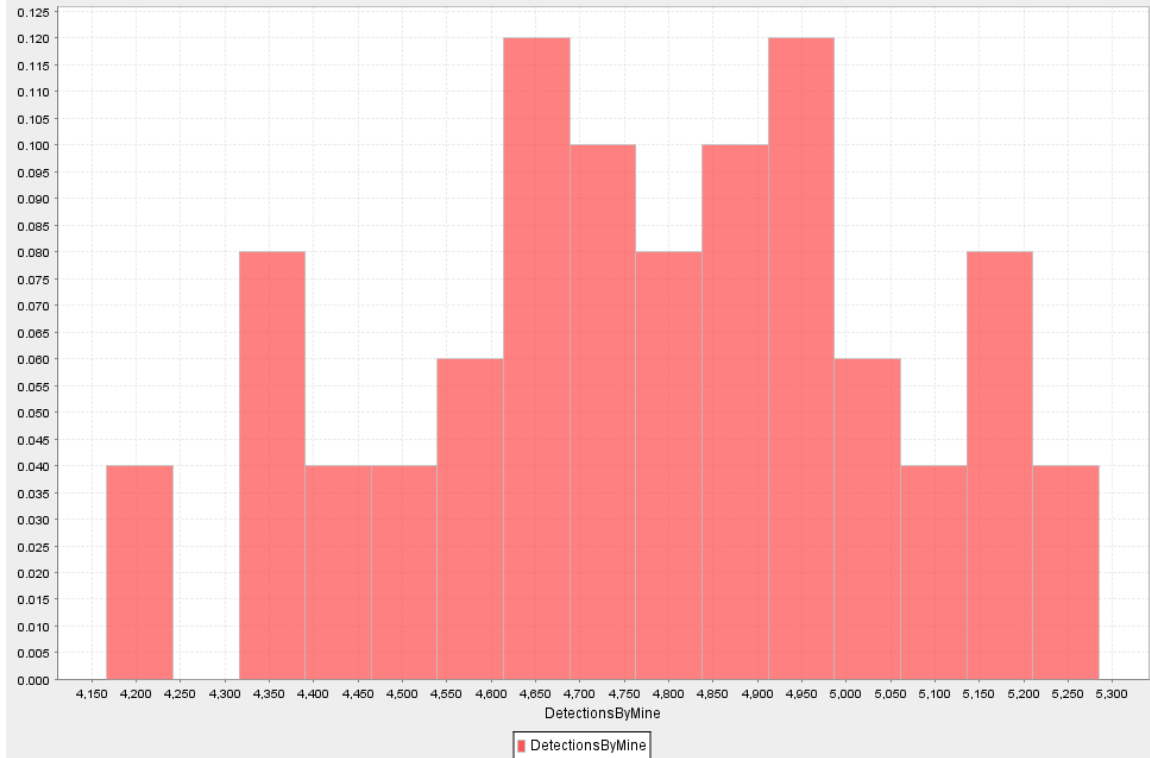
Run#	Count	Min	Max	Mean	StdDev	Variance
1	9860.000	0.000	9859.000	4929.500	2846.481	8102455.000
2	9393.000	0.000	9392.000	4696.000	2711.670	7353153.500
3	10126.000	0.000	10125.000	5062.500	2923.269	8545500.167
4	9559.000	0.000	9558.000	4779.000	2759.590	7615336.667
5	9883.000	0.000	9882.000	4941.000	2853.121	8140297.667
6	10331.000	0.000	10330.000	5165.000	2982.447	8894991.000
7	8849.000	0.000	8848.000	4424.000	2554.631	6526137.500
8	9703.000	0.000	9702.000	4851.000	2801.159	7846492.667
9	9370.000	0.000	9369.000	4684.500	2705.030	7317189.167
10	9828.000	0.000	9827.000	4913.500	2837.244	8049951.000
11	9759.000	0.000	9758.000	4879.000	2817.325	7937320.000
12	9486.000	0.000	9485.000	4742.500	2738.517	7499473.500
13	9532.000	0.000	9531.000	4765.500	2751.796	7572379.667
14	10267.000	0.000	10266.000	5133.000	2963.972	8785129.667
15	9184.000	0.000	9183.000	4591.500	2651.337	7029586.667
16	9770.000	0.000	9769.000	4884.500	2820.500	7955222.500

Run#	Count	Min	Max	Mean	StdDev	Variance
17	9259.000	0.000	9258.000	4629.000	2672.987	7144861.667
18	9675.000	0.000	9674.000	4837.000	2793.076	7801275.000
19	8336.000	0.000	8335.000	4167.500	2406.540	5791436.000
20	10101.000	0.000	10100.000	5050.000	2916.052	8503358.500
21	9717.000	0.000	9716.000	4858.000	2805.201	7869150.500
22	10079.000	0.000	10078.000	5039.000	2909.701	8466360.000
23	8731.000	0.000	8730.000	4365.000	2520.567	6353257.667
24	10122.000	0.000	10121.000	5060.500	2922.114	8538750.500
25	9480.000	0.000	9479.000	4739.500	2736.785	7489990.000
26	8641.000	0.000	8640.000	4320.000	2494.586	6222960.167
27	9708.000	0.000	9707.000	4853.500	2802.603	7854581.000
28	10418.000	0.000	10417.000	5208.500	3007.562	9045428.500
29	10331.000	0.000	10330.000	5165.000	2982.447	8894991.000
30	9348.000	0.000	9347.000	4673.500	2698.679	7282871.000
31	9861.000	0.000	9860.000	4930.000	2846.770	8104098.500
32	8932.000	0.000	8931.000	4465.500	2578.591	6649129.667
33	9950.000	0.000	9949.000	4974.500	2872.462	8251037.500
34	8335.000	0.000	8334.000	4167.000	2406.252	5790046.667
35	9362.000	0.000	9361.000	4680.500	2702.721	7304700.500
36	8773.000	0.000	8772.000	4386.000	2532.691	6414525.167
37	9901.000	0.000	9900.000	4950.000	2858.317	8169975.167
38	10568.000	0.000	10567.000	5283.500	3050.863	9307766.000
39	9267.000	0.000	9266.000	4633.000	2675.297	7157213.000
40	8890.000	0.000	8889.000	4444.500	2566.466	6586749.167
41	8731.000	0.000	8730.000	4365.000	2520.567	6353257.667
42	10345.000	0.000	10344.000	5172.000	2986.489	8919114.167
43	9093.000	0.000	9092.000	4546.000	2625.067	6890978.500
44	9426.000	0.000	9425.000	4712.500	2721.196	7404908.500
45	9092.000	0.000	9091.000	4545.500	2624.779	6889463.000
46	10570.000	0.000	10569.000	5284.500	3051.441	9311289.167
47	9451.000	0.000	9450.000	4725.000	2728.413	7444237.667
48	9352.000	0.000	9351.000	4675.500	2699.834	7289104.667
49	9076.000	0.000	9075.000	4537.500	2620.160	6865237.667
50	9569.000	0.000	9568.000	4784.000	2762.477	7631277.500

Summary Report

Entity	Property	Count	Min	Max	Mean	StdDev	Variance
	DetectionsByMine	50.000	4167.000	5284.500	4773.400	280.909	78909.898
	MineDetections	50.000	87.000	96.000	91.380	2.187	4.781

Conclusions and Recommendations

Conclusions. This simulation was a good exemplar for testing Minefield Search missions for the SeaDiver UUV. For specific conclusions, see Chapter 8.

Recommendations for Future Work. As an exemplar simulation, the behaviors and capabilities of moving entities are very general. More work is required refining the model definitions and search patterns to generate relevant statistics. See Chapter 8 for other specific recommendations for future work.

LIST OF REFERENCES

Buss, A. H. 2004. *Simkit analysis workbench for rapid construction of modeling and simulation components*. Proceedings of the Fall Simulation Interoperability Workshop (September), Accessed 15 August 2006 at http://www.sisostds.org/index.php?tg=fileman&idx=get&id=2&gr=Y&path=Simulation+InAteroperability+Workshops%2F2004+Fall+SIW%2F2004+Fall+SIW+Papers+and+Presentations&file=04F-SIW-020.pdf.

_____. 2002. *Component Based Simulation Modeling With Simkit*. Proceeding of the 2002 Winter Simulation Conference.

———. 2001. *Basic Event Graph Modeling*. Simulation News Europe (April), Accesses 13 March 2007 at http://diana.nps.edu/oa3302/Handouts/BasicEventGraphModeling.doc.

_____. 2000. *The Event List.* OA3302 System Simulation Winter 2007 class notes. Accessed 13 March 2007 at http://diana.nps.edu/oa3302/Handouts/EventList.pdf.

Buss, A. H., and P. J. Sanchez. 2005. *Simple Movement and Detection in Discrete Event Simulation*. Proceeding of the 2005 Winter Simulation Conference. Accessed 13 March 2007 at http://www.informs-cs.org/wsc05papers/118.pdf.

Buss, A. H., and R. Szechtman. 2006. *OA3302 System Simulation*, course notes, Accessed August 2006 at http://diana.nps.edu/oa3302/Handouts/.

Davis, D., and D.P. Brutzman, 2005, "The Autonomous Unmanned Vehicle Workbench: Mission Planning, Mission Rehearsal, and Mission Replay Tool for Physics-based X3D Visualization," *14th International Symposium on Unmanned Untethered Submersible Technology (UUST),* Autonomous Undersea Systems Institute (AUSI), Durham New Hampshire, 21-24 August 2005.

Department of the Navy, *The Navy Unmanned Undersea Vehicle (UUV) Master Plan,* pp. xv-xxv, Government Printing Office, Washington, D.C. 2004.

Dumonteil, R., D. Gassier, J. Rebollo, *Implementing a Low-Cost Long-Range Unmanned Underwater Vehicle: The Seadiver Glider,* report prepared for Naval Postgraduate School, 2006.

Johnson, ADM J. L., USN, and Gen J. L.Jones, USMC. 2000. *U.S. Naval Mine Warfare Plan, 4th Edition, Programs for the New Millennium,* Department of the Navy, Washington, D.C., January.

Goure, D., 2002. Sea-Mine Threat Can No Longer Be Ignored, *National Defense Magazine,* August 2002. Accessed 13 March 2007 at http://www.nationaldefensemagazine.org/issues/2002/Aug/Sea-Mine.htm.

Law, A., and D. Kelton. 2000. *Simulation Modeling and Analysis*, Third Edition, McGraw Hill.

Miller, J.A. *Simulation and Modeling in Java*. Accessed 17 January 2007 at http://chief.cs.uga.edu/~jam/home/courses/csci4210/book.html).

Schruben, L. 1983. Simulation Modeling with Event Graphs. *Communications of the ACM* 26: 957.

Sullivan, P.J., *Evaluating the Effectiveness of Waterside Security Alternatives for Force Protection of Navy Ships and Installations Using X3D Graphics and Agent-Based Simulation*. Master's Thesis, Naval Postgraduate School, Monterey, California, September 2006.

Wu, T. C., 2004. *An Introduction to Object-Oriented Programming with Java*. Third Edition. New York, New York: McGraw-Hill.

INITIAL DISTRIBUTION LIST

1.   Defense Technical Information Center
        Ft. Belvoir, Virginia

2.   Dudley Knox Library
        Naval Postgraduate School
        Monterey, California

3.   Don Brutzman
        Naval Postgraduate School
        Monterey, California

4.   D.C. Boger
        Naval Postgraduate School
        Monterey, California

5.   Curt Blais
        Naval Postgraduate School
        Monterey, California

6.   John Moore
        Navy Modeling and Simulation Office
        Ft. Belvoir, Virginia

7.   John Hiles
        Naval Postgraduate School
        Monterey, California

8.   Arnold Buss
        Naval Postgraduate School
        Monterey, California

9.   Joseph McConnell
        Naval Facilities Engineering Command
        Washington Navy Yard, District of Columbia

10.   CAPT James D. Scola, USN
        Commander, United States Pacific Fleet
        Aiea, Hawaii

11.   Caroline Massie
        Chief of Naval Installations
        Anacostia Annex, District of Columbia

12.   Dr. Brian Donahue
Johns Hopkins University
Baltimore, Maryland

13.   Dr. James D. Miller
Johns Hopkins University
Baltimore, Maryland

14.   CAPT Taylor Skardon
Naval Station Pearl Harbor
Pearl Harbor, Hawaii

15.   Alexandra De Visser
Naval Facilities Engineering Service Center
Port Hueneme California

16.   Dallas Meggitt
Sound and Sea Technology
Edmunds, Washington

17.   Dennis Garrood
Sound and Sea Technology
Edmunds, Washington

18.   Mario Pozzo
Sound and Sea Technology
Edmunds, Washington

19.   Len Daly
Daly Realism
Valley Glen, California

20.   Rick Goldberg
Aniviza Inc.
Los Gatos, California

21.   LT Wilfredo Cruzbaez
Naval Postgraduate School
Monterey, California

22.   Alan Hudson
Yumetech, Inc.
Seattle, Washington

23.   Terry Norbraten
      Naval Postgraduate School
      Monterey, California

24.   CDR John Kennington
      Commander, United States Pacific Fleet
      Aiea, Hawaii

25.   Wendy Walsh
      Naval Postgraduate School
      Monterey, California

26.   Mark W. Kenny
      Center for Submarine Counter-Terrorism Operations
      Groton, Connecticut

27.   CAPT Rick J. Ruehlin
      Littoral and Mine Warfare
      Washington Navy Yard, District of Columbia

28.   Richard L. Snead
      National Security Directorate
      Oak Ridge, Tennessee

29.   Distinguished Professor Anthony Healey
      Department of Mechanical Engineering
      Naval Postgraduate School
      Monterey, California

30.   Dr. Kwang Song
      Department of Mechanical Engineering
      Naval Postgraduate School
      Monterey, California

31.   Douglas Horner
      Department of Mechanical Engineering
      Naval Postgraduate School
      Monterey, California

32.   Sean Krageland
      Department of Mechanical Engineering
      Naval Postgraduate School
      Monterey, California

33. CAPT Dan Gahagan, USN
Naval Research Laboratory
Arlington, Virginia

34. CAPT Dennis Sorensen, USN
Office of Naval Research
Arlington, Virginia

35. Dr. Thomas Swean
Office of Naval Research
Arlington, Virginia

36. Dr. Thomas Curtin
Office of Naval Research
Arlington, Virginia

37. D. Richard Blidberg
Autonomous Undersea Systems Institute
Durham, New Hampshire

38. Steven Chappell
Autonomous Undersea Systems Institute
Durham, New Hampshire

39. Rick Komerska
Autonomous Undersea Systems Institute
Durham, New Hampshire

40. Peter Flynn
Naval Research Laboratory,
Stennis Space Center, Mississippi

41. Mark Falagh
L-3 Communications
Orlando, Florida

42. Eyton Pollach
L-3 Communications
Orlando, Florida

43. Tom Highee
Sound and Sea Technology
Edmunds, Washington

44. Robert Taylor
    Sound and Sea Technology
    Edmunds, Washington

45. Jay Cohen
    Director Science & Technology
    Dept Homeland Security
    Washington, D.C.

46. Dr. Mark Pullen
    Netlab, GMU
    Fairfax, Virginia

47. Dr. Mike Hieb
    Netlab, GMU
    Fairfax, Virginia

48. Erik Chaum
    NUWC
    Newport, Rhode Island

49. David Bellino
    NUWC
    Newport, Rhode Island

50. Pierre Comieau
    NUWC
    Newport, Rhode Island

51. Yvonne Maralawski
    NUWC
    Newport, Rhode Island

52. Virginia Robin Beale
    N81, Pentagon
    Washington, DC

53. LCDR Harrison Schramm
    N81, Pentagon
    Washington, DC

54. LCDR Jeff Debrine
    N81, Pentagon
    Washington, DC

55.	Dr. Jim Eagle
	Operations and Research Department
	Naval Postgraduate School
	Monterey, California

56.	CAPT Jeff Kline, USN (Ret.)
	Operations and Research Department
	Naval Postgraduate School
	Monterey, California

57.	CAPT Wayne Hughes, USN (Ret.)
	Operations and Research Department
	Naval Postgraduate School
	Monterey, California

58.	John Ruch
	Rolands & Associates
	Monterey, California

59.	Dr. R.J. Roland
	Rolands & Associates
	Monterey, California

60.	Ouerghi Nabil
	Moves, NPS
	Monterey, California